ROLLOUT.IO

A REMOTE CONFIGURATION PLATFORM

A PROJECT REPORT

Submitted by

THAKOR PARTHSINH RANVEERSINH [220130116064]

&

ASLALIYA DHARMIK SANJAYBHAI [220130116002]

&

PARMAR MEET NILESHBHAI [220130116036]

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

Department of Computer Engineering and Information Technology

Government Engineering College, Sector 28, Gandhinagar

Gujarat Technological University, Ahmedabad

April, 2026

Government Engineering College

Sector 28, Gandhinagar

CERTIFICATE

This is to certify that the project report submitted along with the project entitled 'ROLLOUT.IO' has been carried out by PARTHSINH THAKOR, DHARMIK ASLALIYA & MEET PARMAR under my guidance in partial fulfilment for the degree of Bachelor of Engineering in Computer Engineering, 8th Semester of Gujarat Technological University, Ahmedabad during the academic year 2025-26.

Prof. Prashant Chaudhari
Internal Guide

Dr. Komal Anadkat
Head of Department

Government Engineering College

Sector 28, Gandhinagar

DECLARATION

We hereby declare that the Internship / Project report submitted along with the Internship / Project entitled ROLLOUT.IO submitted in partial fulfilment for the degree of Bachelor of Engineering in Information Technology to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me / us at Mitra Media Labs under the supervision of Ghanshyam Godhani and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of the Student	Sign of the Student
1. Parthsinh Thakor	_____
2. Dharmik Aslaliya	_____
3. Meet Parmar	_____

Mitra
Media Labs

Certificate Of Internship

Date: 14-Apr-2026

We are delighted to certify that **Mr. Parthsinh R. Thakor(220130116064)** has successfully completed his internship in **Java Backend Developer** at **Mitra Media Labs** from **16 Jan 2026 to 12 Apr 2026.** His outstanding skills, commitment, dedication and innovative contributions were highly valued.

During this tenure, we found him to be sincere and hardworking. He displayed a lot of initiative to learn and improve upon the service standards of our organization. He is well mannered and pleasant to move with.

Mitra Media Labs offers complete cycle of IT service in the domain of Java Backend Development, Web-based Technologies, Market Solutions, Internet marketing and Web Solutions.

We wish him a very best of luck in his future endeavours.

Regards,
MITRA MEDIA LABS

FOR,MITRA MEDIA LABS
G. M. Patel
PROPRIETOR

Director
(Ghanshyam Godhani)

Web: www.mimelabs.net +91 9428333211 **Email**: hello@mimelabs.net

Mitra
Media Labs

Certificate Of Internship

Date: 14-Apr-2026

We are delighted to certify that **Mr. Dharmik S. Aslaliya(220130116002)** has successfully completed his internship in **Java Backend Developer** at **Mitra Media Labs** from **16 Jan 2026 to 12 Apr 2026.** His outstanding skills, commitment, dedication and innovative contributions were highly valued.

During this tenure, we found him to be sincere and hardworking. He displayed a lot of initiative to learn and improve upon the service standards of our organization. He is well mannered and pleasant to move with.

Mitra Media Labs offers complete cycle of IT service in the domain of Java Backend Development, Web-based Technologies, Market Solutions, Internet marketing and Web Solutions.

We wish him a very best of luck in his future endeavours.

Regards,
MITRA MEDIA LABS

FOR,MITRA MEDIA LABS
G. M. Patel.
PROPRIETOR.

Director
(Ghanshyam Godhani)

Web: www.mimelabs.net +91 9428333211 **Email**: hello@mimelabs.net

Mitra
Media Labs

Certificate Of Internship

Date: 14-Apr-2026

We are delighted to certify that **Mr. Meet N. PARMAR(220130116036)** has successfully completed his internship in **Frontend Developer** at **Mitra Media Labs** from **16 Jan 2026 to 10 Apr 2026**. His outstanding skills, commitment, dedication and innovative contributions were highly valued.

During this tenure, we found him to be sincere and hardworking. He displayed a lot of initiative to learn and improve upon the service standards of our organization. He is well mannered and pleasant to move with.

Mitra Media Labs offers complete cycle of IT service in the domain of Frontend Development, Web-based Technologies, Market Solutions, Internet marketing and Web Solutions.

We wish him a very best of luck in his future endeavours.

Regards,
MITRA MEDIA LABS

FOR,MITRA MEDIA LABS
G.M. Patel
PROPRIETOR

Director
(Ghanshyam Godhani)

Web: www.mimelabs.net +91 9428333211 **Email**: hello@mimelabs.net

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **Mr. Ghanshyam Godhani**, Project Manager of Mitra Media Labs, for giving us the opportunity to work as interns in their organization. His guidance and support throughout the internship were extremely valuable.

We would also like to express our heartfelt appreciation to all the members of Mitra Media Labs for their continuous support and cooperation during the internship period.

We would like to extend our gratitude to the Head of the Department, **Dr. Komal Anadkat**, the Internship Coordinator of the IT Department, **Prof. C.M. Kapadiya**, and our Internal Guide, **Prof. Prashant Chaudhari**, for their constant support, valuable guidance, and encouragement throughout the internship.

Finally, we would like to thank all the faculty members, staff, and our friends who directly or indirectly helped us in the successful completion of this internship.

Parthsinh Thakor 220130116064
Dharmik Aslaliya 220130116002
Meet Parmar 220130116036

ABSTRACT

This report presents the work carried out during a 12-week internship at Mitra Media Labs. The objective of the internship was to gain practical exposure to real-world software development using modern technologies.

During the internship, we worked on the project titled “ROLLOUT.IO: A Remote Configuration Platform.” The project aims to develop a centralized system that allows dynamic control of application features and configurations without requiring redeployment. It enables developers to manage feature flags and control application behaviour in real time.

The system is built using a microservices architecture for scalability and flexibility. The backend includes components such as API Gateway, Service Discovery, and Authentication Service, while the frontend provides a user-friendly interface for managing configurations and feature flags.

Through this internship, we gained hands-on experience in both frontend and backend development, microservices design, and system architecture, along with valuable insights into industry practices and teamwork.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
API	Application Programming Interface
UI	User Interface
SDK	Software Development Kit
JWT	JSON Web Token
JSON	JavaScript Object Notation
DB	Database
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
URL	Uniform Resource Locator
MVC	Model View Controller
RBAC	Role-Based Access Control
CI/CD	Continuous Integration / Continuous Deployment
TTL	Time To Live
CPU	Central Processing Unit
JVM	Java Virtual Machine
APM	Application Performance Monitoring
POC	Proof of Concept
GUI	Graphical User Interface

CHAPTER 1: OVERVIEW OF THE COMPANY

1.1: HISTORY

Mitra Media Labs is a privately held software development and IT consulting company based in Surat, Gujarat. The company focuses on delivering digital transformation solutions and helping businesses grow using modern technologies.

The company is led by Ghanshyam Godhani, who plays a key role in driving innovation, maintaining a strong work culture, and ensuring the delivery of high-quality technology solutions.

Since its establishment, Mitra Media Labs has expanded its services to cater to startups, digital agencies, and enterprises. It provides expertise in web and mobile application development, UI/UX design, frontend development, cloud solutions, and DevOps.

1.2: SCOPE OF WORK

Mitra Media Labs operates in the field of software development and IT consulting, providing innovative and scalable digital solutions to businesses.

The organization works with startups, digital agencies, and enterprises to deliver customized solutions that enhance business efficiency and digital presence. It follows a client-centric approach and emphasizes quality, scalability, and performance in all its projects.

The major sectors served by the company include:

- Startups and SaaS platforms
- E-commerce and digital businesses
- Enterprise and corporate systems
- Service-based and on-demand applications
- IT and software development industry
- Cloud-based and technology-driven solutions

1.3: SERVICES

Mitra Media Labs provides a wide range of software development and IT consulting services to meet the evolving needs of businesses. The company focuses on delivering high-quality, scalable, and efficient digital solutions using modern technologies and development practices.

The services are designed to help clients improve their digital presence, streamline operations, and build reliable software systems tailored to their requirements.

The key services offered by the company include:

- Web application development
- Mobile application development
- Frontend and backend development
- UI/UX design and prototyping
- API development and integration
- Cloud-based solutions and deployment
- DevOps and maintenance services
- Microservices architecture development

CHAPTER 2: ORGANIZATION STRUCTURE

2.1: DETAILS OF WORK CARRIED OUT IN DEPARTMENT

Mitra Media Labs is organized into various departments that work collaboratively to deliver high-quality software solutions.

Project Management Department: This department is responsible for planning, organizing, and managing projects. It ensures proper communication between clients and development teams, defines project requirements, and monitors progress to meet deadlines.

Development Department: The development department handles both frontend and backend development of applications. It focuses on designing, coding, and implementing software solutions using modern technologies and frameworks.

UI/UX Design Department: This department is responsible for designing user interfaces and improving user experience. It works on creating visually appealing and user-friendly designs for web and mobile applications.

Quality Assurance (QA) Department: The QA department ensures that the developed software meets quality standards. It performs testing, identifies bugs, and validates system functionality before deployment.

DevOps/Deployment Department: This department manages deployment, maintenance, and monitoring of applications. It focuses on automation, cloud services, and continuous integration and deployment practices.

2.2: LIST THE TECHNICAL SPECIFICATIONS OF MAJOR EQUIPMENT USED IN EACH DEPARTMENT

Mitra Media Labs utilizes modern computing systems and software tools across different departments to ensure efficient software development and deployment. The technical specifications of major equipment used in each department are as follows:

Development Department:

- Systems with Intel i5/i7 processors or equivalent
- Minimum 8GB–16GB RAM
- SSD storage (256GB or above)
- Development tools: VS Code, IntelliJ IDEA
- Technologies: Java, Spring Boot, Node.js, React.js
- Databases: MySQL, MongoDB

UI/UX Design Department:

- High-performance systems with graphic support
- Design tools: Figma, Adobe XD
- Software for prototyping and interface design

Quality Assurance (QA) Department:

- Systems with standard configurations (8GB RAM minimum)
- Testing tools: Postman, Selenium (basic usage)
- Browsers for cross-platform testing

DevOps / Deployment Department:

- Systems configured for server management and deployment
- Tools: Git, Docker
- Cloud platforms: AWS or similar services
- CI/CD tools for automation

Project Management Department:

- Systems with standard configurations
- Tools: Jira, Trello, or similar project tracking tools
- Communication tools: Slack, Email, Google Meet

2.3 SCHEMATIC LAYOUT SHOWING SEQUENCE OF OPERATION

Requirement Analysis: At Mitra Media Labs, the process begins with understanding client requirements and business needs. The project team communicates with clients to gather detailed specifications and define the scope of the software solution.

System Design: After requirement analysis, the design phase is carried out where system architecture, database structure, and overall workflow are planned. The team ensures that the design supports scalability, flexibility, and performance.

Development: In the development phase, the application is built using modern technologies. The backend team works on business logic, APIs, and data management, while the frontend team develops user interfaces to ensure a smooth and interactive user experience.

Integration: Different modules and services are integrated to ensure proper communication within the system. APIs and microservices are connected to build a complete and functional application.

Testing and Quality Assurance: The developed system undergoes testing to identify bugs and ensure quality. The QA team verifies functionality, performance, and reliability before deployment.

Deployment: Once testing is completed, the application is deployed on cloud or server environments. The deployment process ensures that the system is accessible and performs efficiently in a real-world environment.

Maintenance and Support: After deployment, continuous monitoring and maintenance are performed.

CHAPTER 3: INTRODUCTION TO PROJECT

3.1: PROJECT SUMMARY

The internship was carried out at Mitra Media Labs for a duration of 12 weeks with the objective of gaining practical exposure to real-world software development. During this period, we worked on a project titled "ROLLOUT.IO: A Remote Configuration Platform."

The main aim of the project was to develop a centralized system that enables dynamic control of application features and configurations without requiring redeployment. The platform allows developers to manage feature flags, control application behaviour in real time, and perform gradual feature rollouts efficiently.

The system was developed using a microservices architecture to ensure scalability, flexibility, and maintainability. The backend services were implemented to handle business logic, API management, authentication, and data processing, while the frontend was designed to provide a user-friendly interface for managing configurations and feature flags.

Throughout the internship, we gained hands-on experience in both frontend and backend development, API integration, and system design. We also learned about industry practices such as version control, deployment strategies, and teamwork, which helped us enhance our technical and professional skills.

3.2: PURPOSE

The purpose of this internship project is to design and develop a centralized remote configuration platform that allows dynamic control over application features without requiring redeployment. The project aims to simplify feature management and improve flexibility in software applications.

Another purpose of the project is to implement a scalable and maintainable system using microservices architecture, enabling efficient handling of multiple services and smooth integration between components.

3.3: OBJECTIVE

The main objectives of the internship project are as follows:

- To design and develop a remote configuration platform for dynamic control of application features
- To implement feature flag functionality for enabling and disabling features without redeployment
- To build a scalable and maintainable system using microservices architecture
- To develop secure and efficient backend services for API management and data handling
- To design a user-friendly frontend interface for managing configurations and feature flags
- To integrate different system components using APIs and ensure smooth communication
- To gain practical knowledge of frontend and backend development using modern technologies
- To understand industry practices such as version control, testing, and deployment

3.4: SCOPE

The scope of the project includes the design and development of a remote configuration platform that enables dynamic control of application features. The system allows users to manage feature flags, update configurations in real time, and control application behaviour without requiring redeployment.

The project covers the development of both backend and frontend components. The backend is responsible for handling business logic, API development, authentication,

and data management, while the frontend provides a user-friendly interface for managing configurations and feature flags.

The system is designed using a microservices architecture to ensure scalability, flexibility, and maintainability. It supports integration between different services and provides a structured approach for managing configurations across applications.

3.5: TECHNOLOGY AND LITERATURE REVIEW

3.5.1 Technology Used

The project “ROLLOUT.IO: A Remote Configuration Platform” is developed using modern technologies to ensure scalability, performance, and maintainability. The backend is developed using Java and Spring Boot framework, which provides a robust environment for building microservices-based applications. It supports RESTful API development, security, and efficient handling of business logic.

The frontend is developed using standard web technologies such as HTML, CSS, and JavaScript, along with modern frameworks to create a responsive and user-friendly interface. The system uses MySQL and MongoDB databases for efficient data storage and retrieval.

3.5.2 Literature Review

The concept of remote configuration and feature flag systems has become increasingly important in modern software development. Various studies and industry practices highlight the need for dynamic control of application features without requiring redeployment. Feature flags enable developers to release features gradually, perform A/B testing, and reduce risks during deployment.

Modern software systems commonly adopt microservices architecture to achieve scalability and flexibility. Research shows that microservices allow independent deployment of services, better fault isolation, and improved system maintainability.

Many organizations use remote configuration platforms to manage application behaviour in real time and enhance user experience.

The project is inspired by these concepts and aims to implement a simplified version of such systems by combining feature flag management with microservices architecture, providing a scalable and efficient solution.

3.6: PROJECT / INTERNSHIP PLANNING

Requirement Analysis Stage: In this stage, the project requirements were identified and analysed. The team understood the objectives and functionalities of the system based on the project idea.

Design Stage: After requirement analysis, the system design was prepared. This included defining the architecture, database structure, and overall workflow of the application.

Development Stage: During this stage, the system was implemented by developing both frontend and backend components. APIs were created, and different modules were developed according to the design.

Integration Stage: In this stage, different modules and services were integrated to ensure proper communication between components and smooth functioning of the system.

Testing Stage: The developed system was tested to identify and fix errors. This ensured that the application met all functional requirements and worked correctly.

Deployment Stage: After successful testing, the system was deployed on a server or cloud platform, making it accessible for users.

Maintenance Stage: Finally, the system was monitored and maintained to ensure performance, reliability, and continuous improvement.

Project Development Approach and Justification

The project follows a microservices-based development approach to ensure scalability, flexibility, and maintainability. In this approach, the system is divided into multiple independent services such as authentication, configuration management, SDK service, config server, registry server and API gateway.

This approach is chosen because it allows independent development and deployment of services, improves fault isolation, and makes the system easier to scale. It also supports better organization of code and efficient handling of complex applications.

Project Effort and Time Estimation

The project was planned over a duration of 12 weeks, with different phases allocated specific time periods.

- Requirement Analysis and Planning: 1–2 weeks
- System Design: 1–2 weeks
- Development (Frontend and Backend): 5–6 weeks
- Testing and Debugging: 2 weeks
- Deployment and Finalization: 1 week

This estimation helped in managing time effectively and ensuring completion of the project within the internship duration.

Roles and Responsibility

The project was carried out by a team of three members, with each member assigned specific roles and responsibilities.

- Backend Development: Responsible for API development, business logic, and database management
- Frontend Development: Responsible for designing user interfaces and integrating with backend services
- Integration and Testing: Responsible for combining modules, testing functionalities, and ensuring system performance

All team members collaborated during planning, development, and problem-solving to ensure successful completion of the project.

Group Dependencies

The project involved interdependencies between different modules and team members. The frontend depended on backend APIs for data, while backend development relied on proper database design and system architecture.

Similarly, testing depended on the completion of development modules, and deployment required successful integration of all components. Effective communication and coordination among team members ensured smooth handling of these dependencies and timely progress of the project.

3.7: PROJECT/INTERNSHIP SCHEDULING

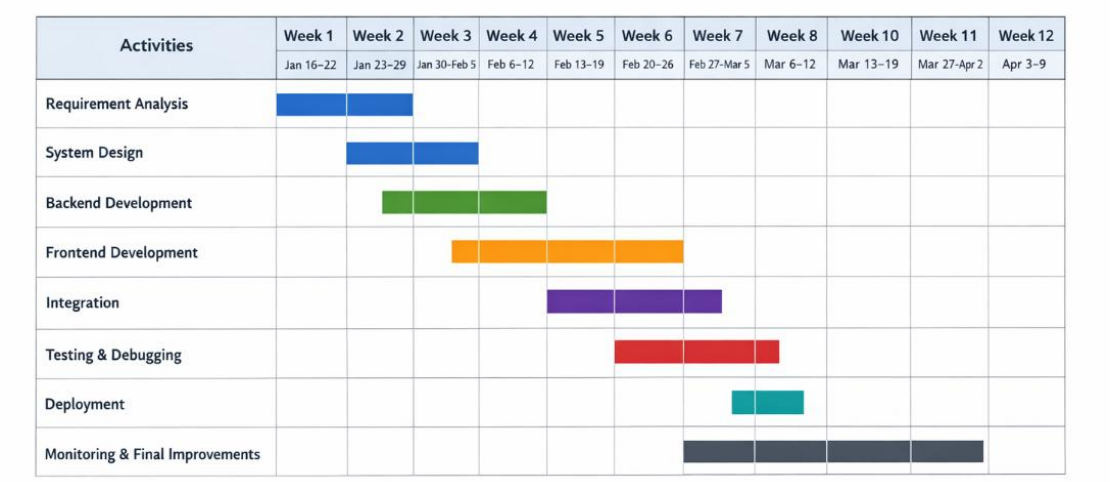

Fig-3.1: Gantt Chart

The Gantt chart illustrates the project schedule over a period of 12 weeks, from January 16 to April 13. It represents the sequence and duration of various project activities, including requirement analysis, system design, development, integration, testing, and deployment.

Each activity is allocated a specific time frame, showing how different phases of the project are organized and executed. Some activities, such as frontend and backend development, are carried out in parallel to optimize time and improve efficiency.

The chart provides a clear visualization of task distribution, progress, and dependencies, helping in effective planning, coordination, and timely completion of the project.

3.8 DEVELOPMENT METHODOLOGY

The Rollout.io system was developed using an Agile-based development methodology. This approach focuses on iterative development, continuous improvement, and flexibility in handling changing requirements.

Agile Approach: The development process was divided into multiple small iterations, where each module of the system was designed, implemented, and tested incrementally. This allowed gradual enhancement of the system and early identification of issues.

Phases Followed: The following phases were followed during development:

- **Requirement Analysis:** Understanding the need for a feature flag management system and identifying key functionalities such as feature toggling, targeting rules, and SDK integration.
- **System Design:** Designing the microservices architecture including Auth Service, Control Plane Service, SDK Service, and API Gateway.
- **Implementation:** Developing backend services using Spring Boot, frontend dashboard using React, and SDK for client integration.
- **Testing:** Performing API testing, UI testing, and real-time validation using feature flag updates.
- **Iteration and Improvement:** Continuously improving the system by adding advanced features such as dependency management, JSON configurations, and monitoring.

Advantages of Chosen Methodology: Advantages are given below:

- Provides flexibility to adapt to changing requirements
- Enables incremental development and testing
- Helps in early detection and resolution of issues
- Supports continuous improvement of the system

CHAPTER 4: SYSTEM ANALYSIS

4.1 PROBLEM STATEMENT

In modern software applications, managing features and configurations dynamically is a major challenge. Most traditional systems rely on code-level changes and redeployment whenever any modification is required in application behaviour. This makes the system rigid, time-consuming, and inefficient for handling frequent updates.

There is a lack of centralized control for managing features across different environments. Developers are unable to enable or disable features in real time, which limits flexibility and slows down the development and release process. Additionally, existing systems do not support gradual feature rollout, user-based targeting, or A/B testing effectively.

Furthermore, handling dependencies between different features becomes complex in large-scale applications. Without a proper mechanism, this can lead to inconsistent behaviour and increased chances of errors. Therefore, there is a need for a scalable and efficient system that allows dynamic feature management, real-time control, and improved flexibility without requiring redeployment.

4.2 STUDY OF CURRENT SYSTEM

In the existing software development environment, application features and configurations are primarily managed through static methods such as hardcoded values or configuration files. Any change in feature behaviour requires modification in the source code followed by recompilation and redeployment of the application.

Most systems lack a centralized platform for managing configurations across multiple environments. Feature control is often handled manually, making it difficult to maintain consistency and scalability in large applications. Developers have limited ability to control feature visibility once the application is deployed.

Additionally, real-time updates and dynamic configuration changes are not supported effectively in traditional systems. There is no proper mechanism for gradual feature rollout, user-based targeting, or experimentation techniques such as A/B testing.

4.3 PROBLEMS / WEAKNESSES OF CURRENT SYSTEM

The existing system for managing application features and configurations has several limitations, which affect flexibility, scalability, and efficiency.

- **Dependence on Redeployment:** Any change in feature behaviour requires code modification and redeployment, which is time-consuming and inefficient.
- **Lack of Real-Time Control:** Features cannot be enabled or disabled dynamically, making it difficult to respond quickly to changing requirements.
- **No Centralized Management:** There is no unified platform to manage features across different environments, leading to inconsistency and complexity.
- **Limited Scalability:** Traditional approaches are not suitable for large-scale applications with multiple services and teams.
- **No Support for Gradual Rollout:** Features cannot be released gradually or to specific user groups, limiting testing and experimentation capabilities.
- **Poor Handling of Dependencies:** Managing dependencies between features is difficult, which may lead to inconsistent system behaviour.
- **Increased Risk of Errors:** Frequent deployments increase the chances of introducing bugs and system instability.

4.4 REQUIREMENTS OF NEW SYSTEM

To overcome the limitations of the current system, the proposed system should fulfil the following requirements:

- **Dynamic Feature Management:** The system should allow enabling or disabling features in real time without requiring redeployment.
- **Centralized Configuration Control:** A centralized platform should be available to manage feature flags and configurations across different environments.

- **Scalability:** The system should support large-scale applications and handle multiple services efficiently using a scalable architecture.
- **User-Based Targeting:** It should allow feature control based on user attributes, enabling personalized feature delivery.
- **Gradual Feature Rollout:** The system should support percentage-based rollout and controlled release of features.
- **Dependency Management:** Proper handling of dependencies between features should be provided to ensure consistent behaviour.
- **Security and Authentication:** Secure access control mechanisms should be implemented to protect system data and operations.
- **High Performance:** The system should provide fast response times using caching and optimized data handling.
- **Reliability and Maintainability:** The system should be stable, easy to maintain, and support continuous updates without affecting performance.

4.5 PROPOSED SYSTEM

To overcome the limitations of the existing system, a new system named "ROLLOUT.IO: A Remote Configuration Platform" is proposed. This system is designed to provide dynamic control over application features and configurations without requiring redeployment.

The proposed system follows a microservices-based architecture, where different services such as authentication, configuration management, API gateway, and service registry work together to provide a scalable and flexible solution. Each service is independently deployable and communicates through APIs, ensuring better modularity and maintainability.

The system allows developers to create and manage feature flags, enabling or disabling features in real time. It also supports gradual feature rollout, user-based targeting, and efficient configuration management across multiple environments.

A centralized dashboard is provided for users to control and monitor feature flags and configurations. The system ensures secure access through authentication mechanisms and provides fast performance using optimized data handling techniques.

4.6 SYSTEM FLOW

4.6.1 Use Case Diagram of the System

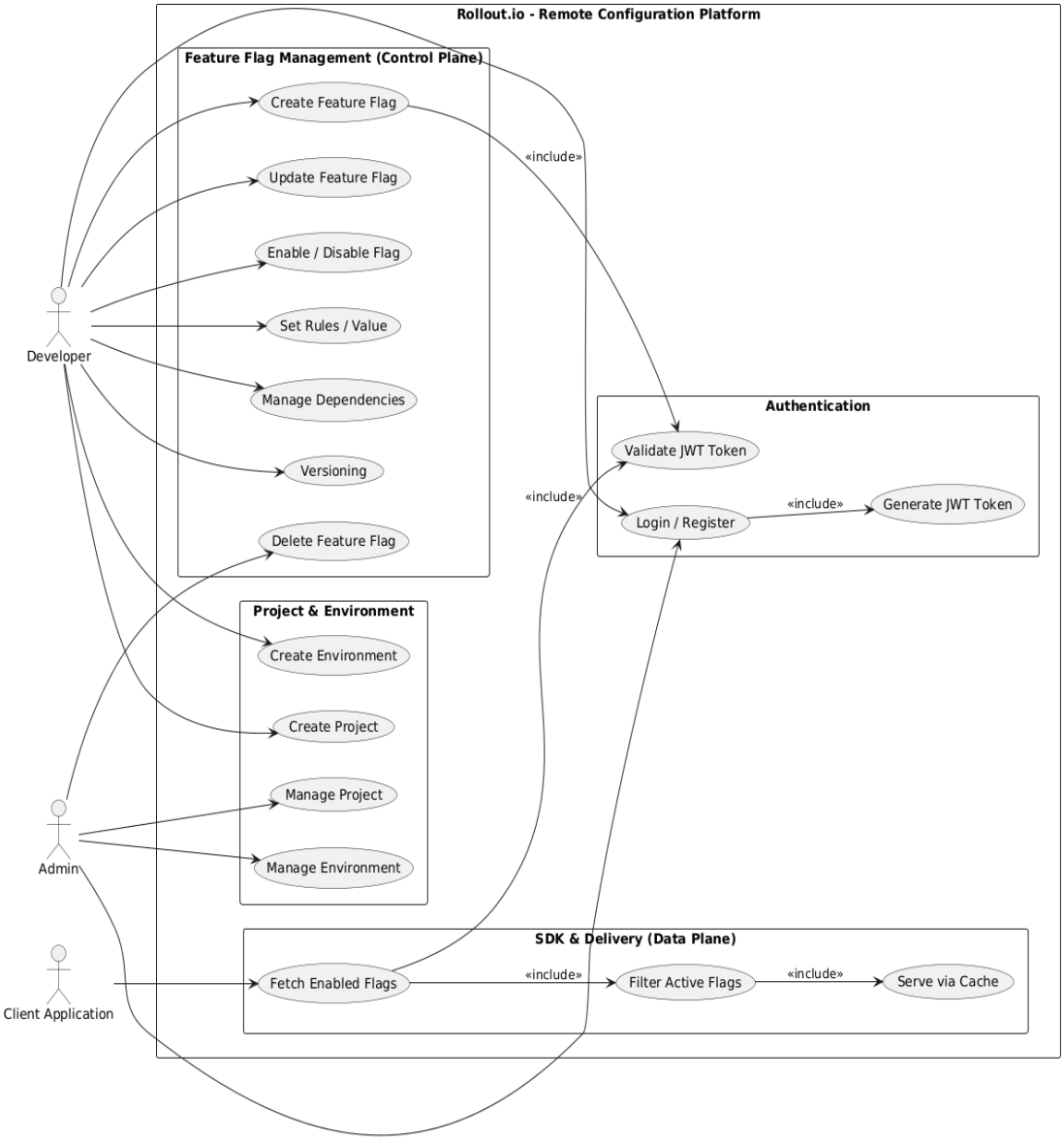

Fig – 4.1: Use Case Diagram

4.6.2 Activity Diagram of the System

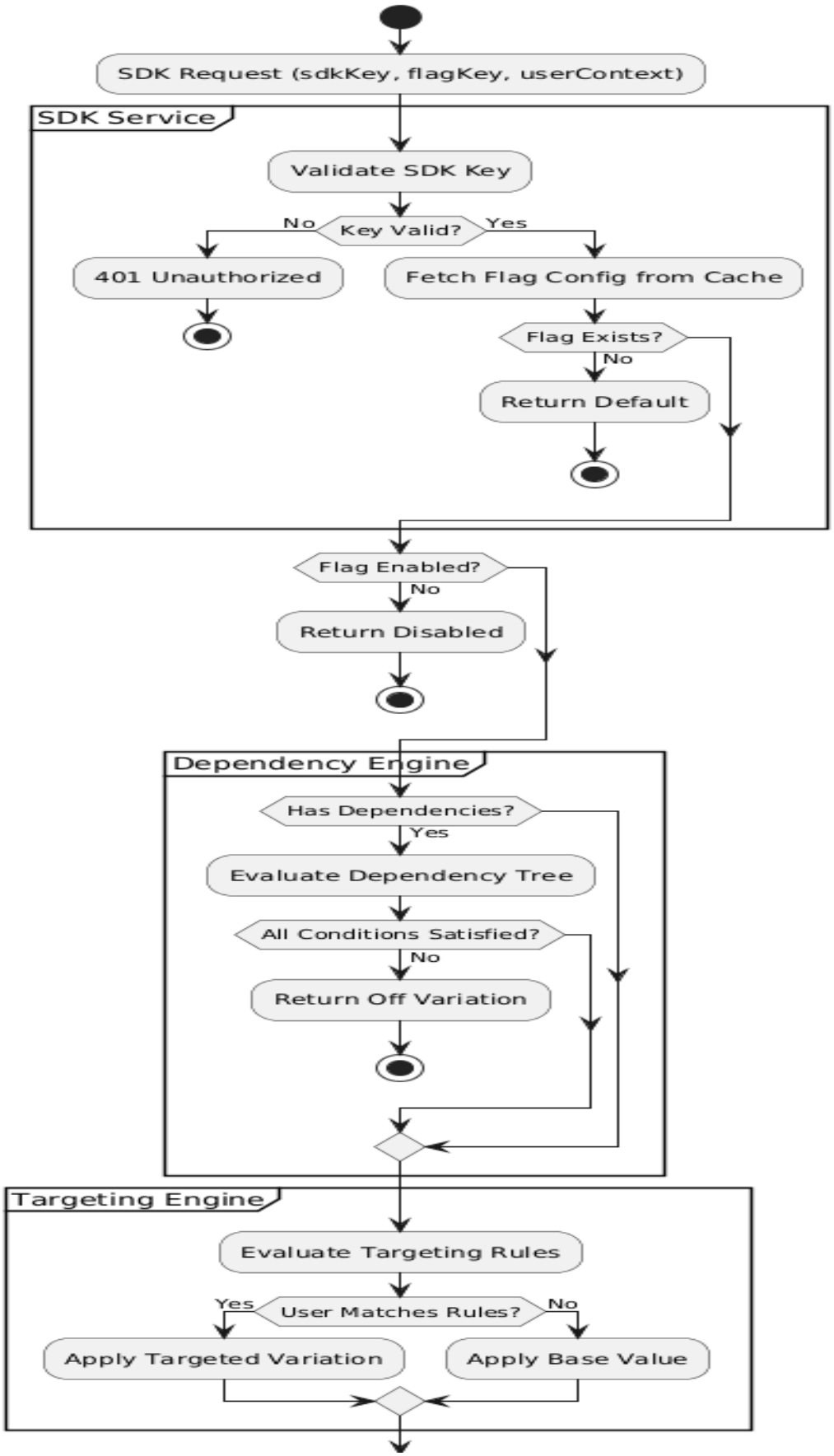

Fig – 4.2: Activity Diagram

4.6.3 Authentication Sequence Diagram

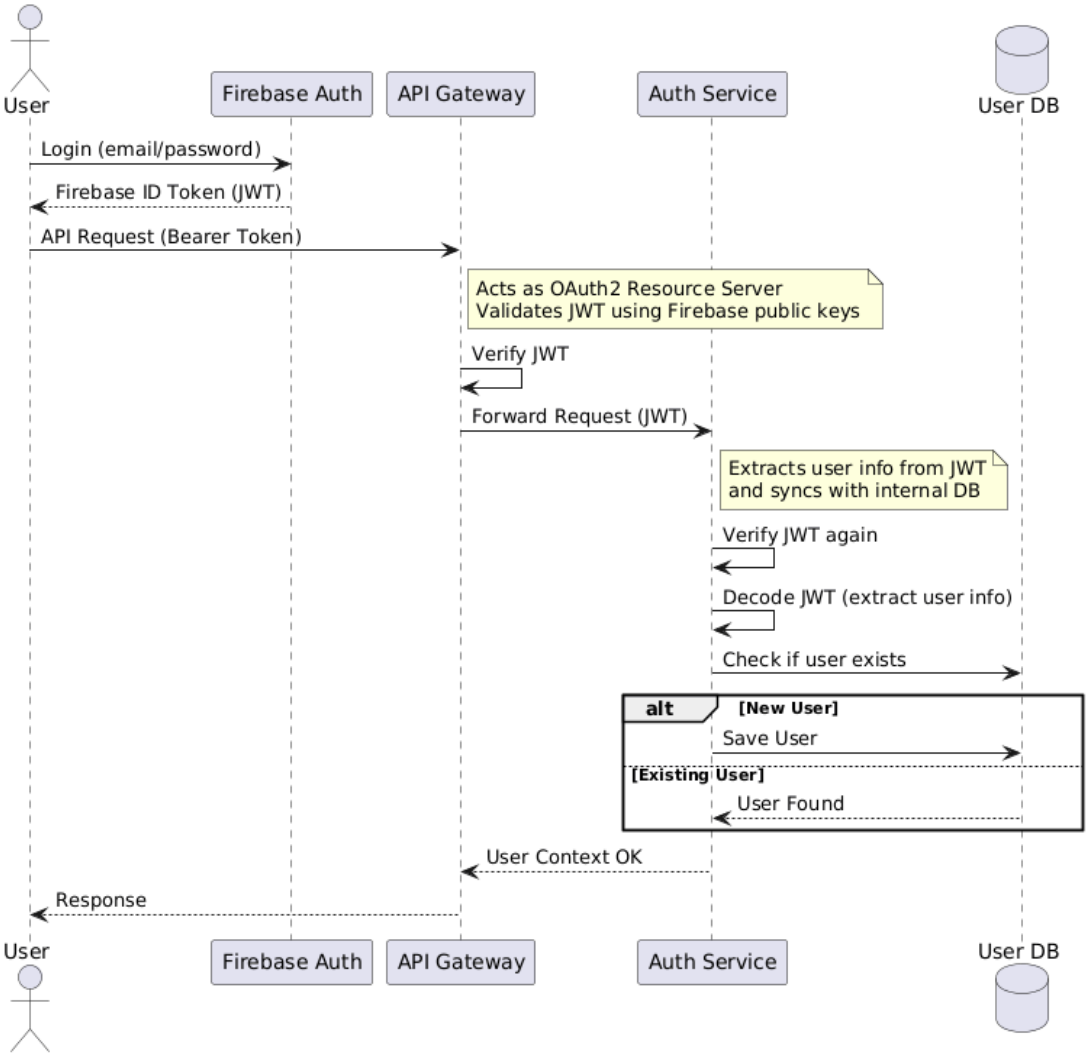

Fig – 4.3: Authentication Sequence Diagram

4.6.4 Feature Flag Creation Sequence Diagram

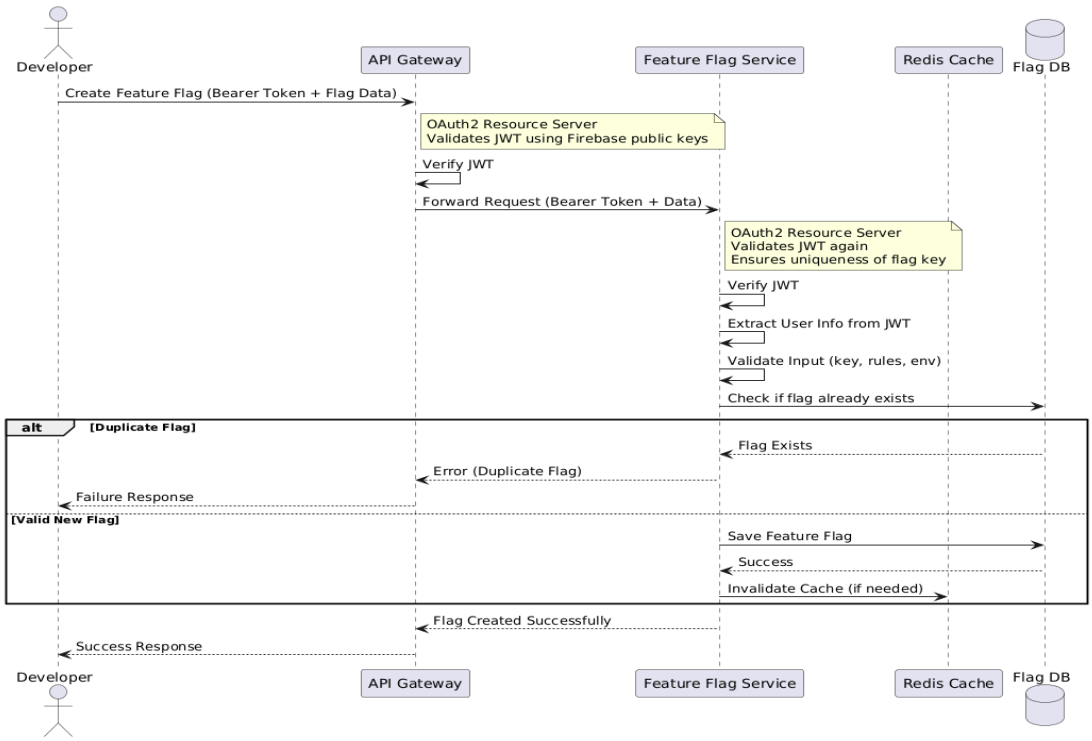

Fig – 4.4: Feature Flag Management Sequence Diagram

4.6.5 SDK Sequence Diagram

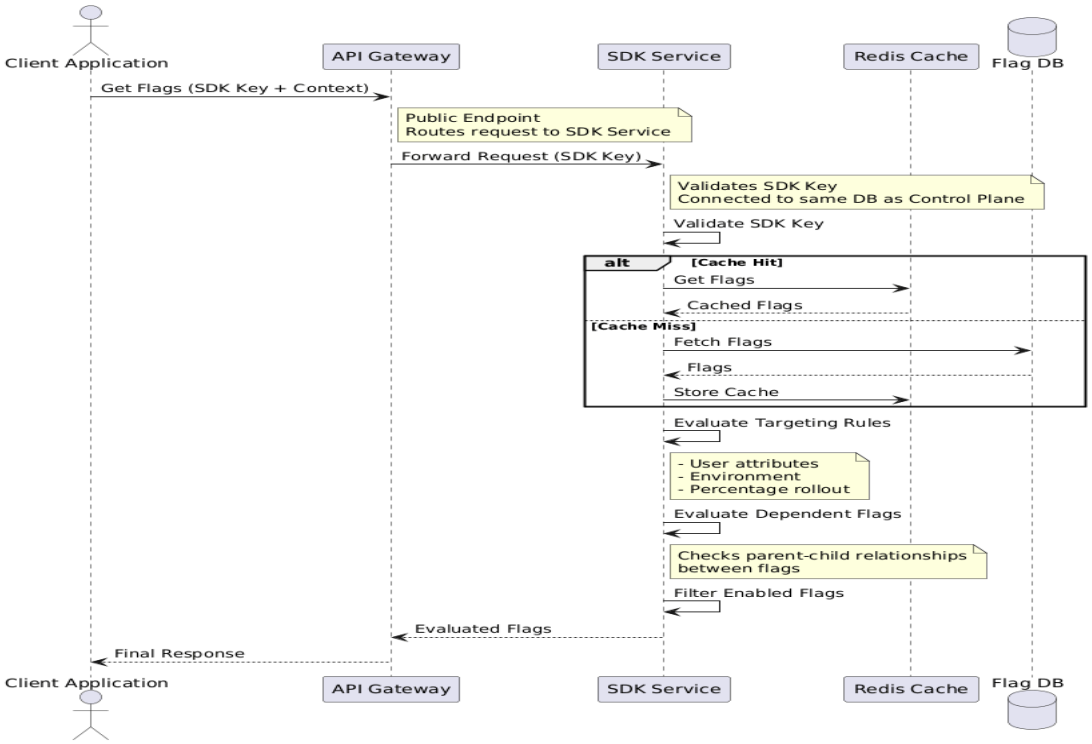

Fig – 4.5: SDK Sequence Diagram

4.6.6 State Diagram of the System

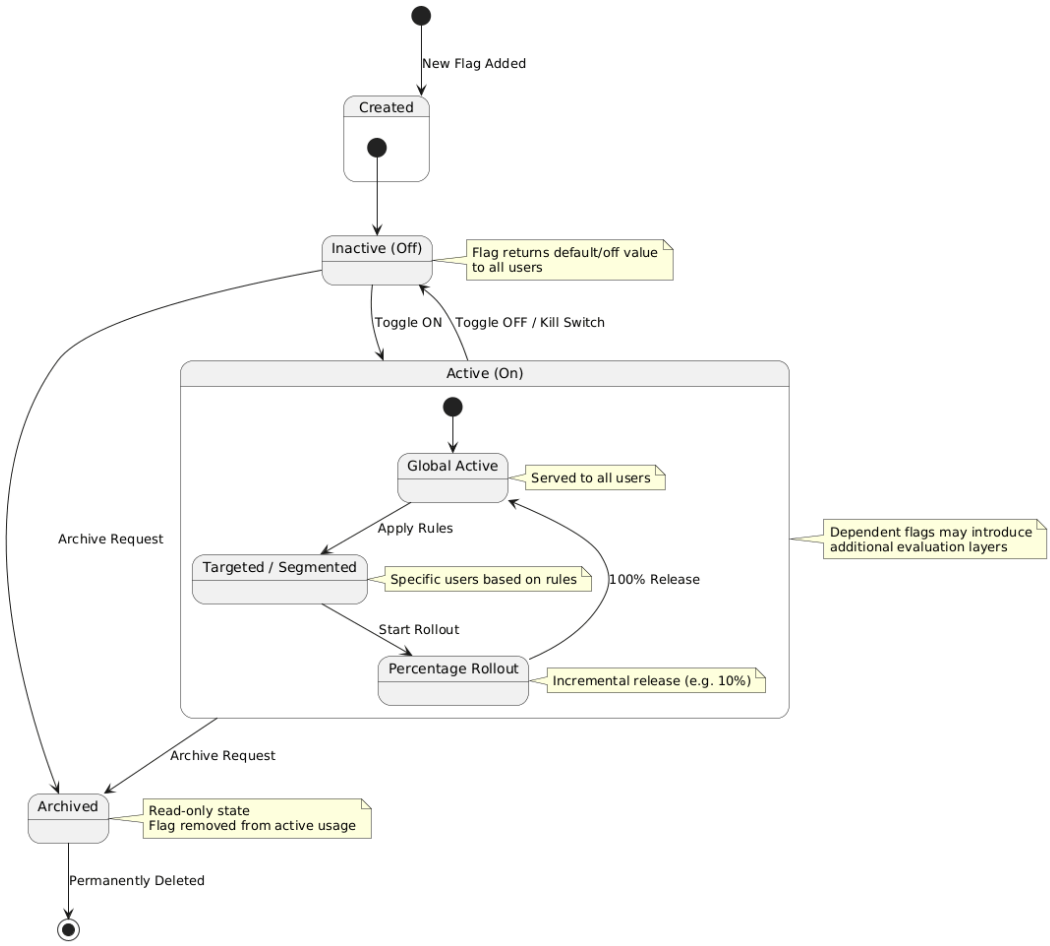

Fig – 4.6: State Diagram

4.7 FEATURES OF PROPOSED SYSTEM

The proposed system "ROLLOUT.IO: A Remote Configuration Platform" provides several advanced features to improve flexibility, scalability, and efficiency in managing application configurations.

- **Dynamic Feature Control:** Enables real-time activation or deactivation of features without redeployment.
- **Centralized Management:** Provides a single dashboard to manage feature flags and configurations across multiple environments.
- **Microservices Architecture:** Ensures scalability, flexibility, and independent deployment of system components.

- **User-Based Targeting:** Allows features to be enabled for specific users based on attributes and conditions.
- **Gradual Rollout:** Supports percentage-based rollout for controlled feature release.
- **Dependency Management:** Handles relationships between feature flags to ensure consistent system behaviour.
- **High Performance with Caching:** Uses caching mechanisms (Redis) to reduce latency and improve response time.
- **Secure Authentication:** Implements JWT-based authentication for secure access control.
- **Real-Time Updates:** Allows instant updates to configurations without affecting running systems.
- **Monitoring and Observability:** Integrates monitoring tools to track system performance and logs.

CHAPTER 5: SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The system architecture of "ROLLOUT.IO: A Remote Configuration Platform" is designed using a microservices-based approach to ensure scalability, flexibility, and maintainability. In this architecture, the system is divided into multiple independent services, each responsible for a specific functionality.

The main components of the system include the API Gateway, Authentication Service, Feature Flag (Control Plane) Service, SDK Service (Data Plane), Config Server, and Service Registry. The API Gateway acts as a single-entry point for all client requests and routes them to the appropriate services. The Authentication Service handles user authentication and authorization using JWT tokens.

The Control Plane Service is responsible for managing feature flags, projects, and environments, while the SDK Service handles flag evaluation and provides feature data to client applications. The Config Server manages external configurations, and the Service Registry (Eureka) enables communication between microservices.

This architecture allows independent deployment of services, better fault isolation, and efficient scaling of individual components. It also ensures that the system can handle high traffic and provide real-time responses.

5.1.1 System Design Diagram

The System Design Diagram represents the complete architectural structure of the Rollout.io platform and illustrates how different components of the system interact with each other. The system is designed using a microservices-based architecture, which allows independent development, deployment, and scaling of individual services. This approach improves system flexibility, fault tolerance, and maintainability.

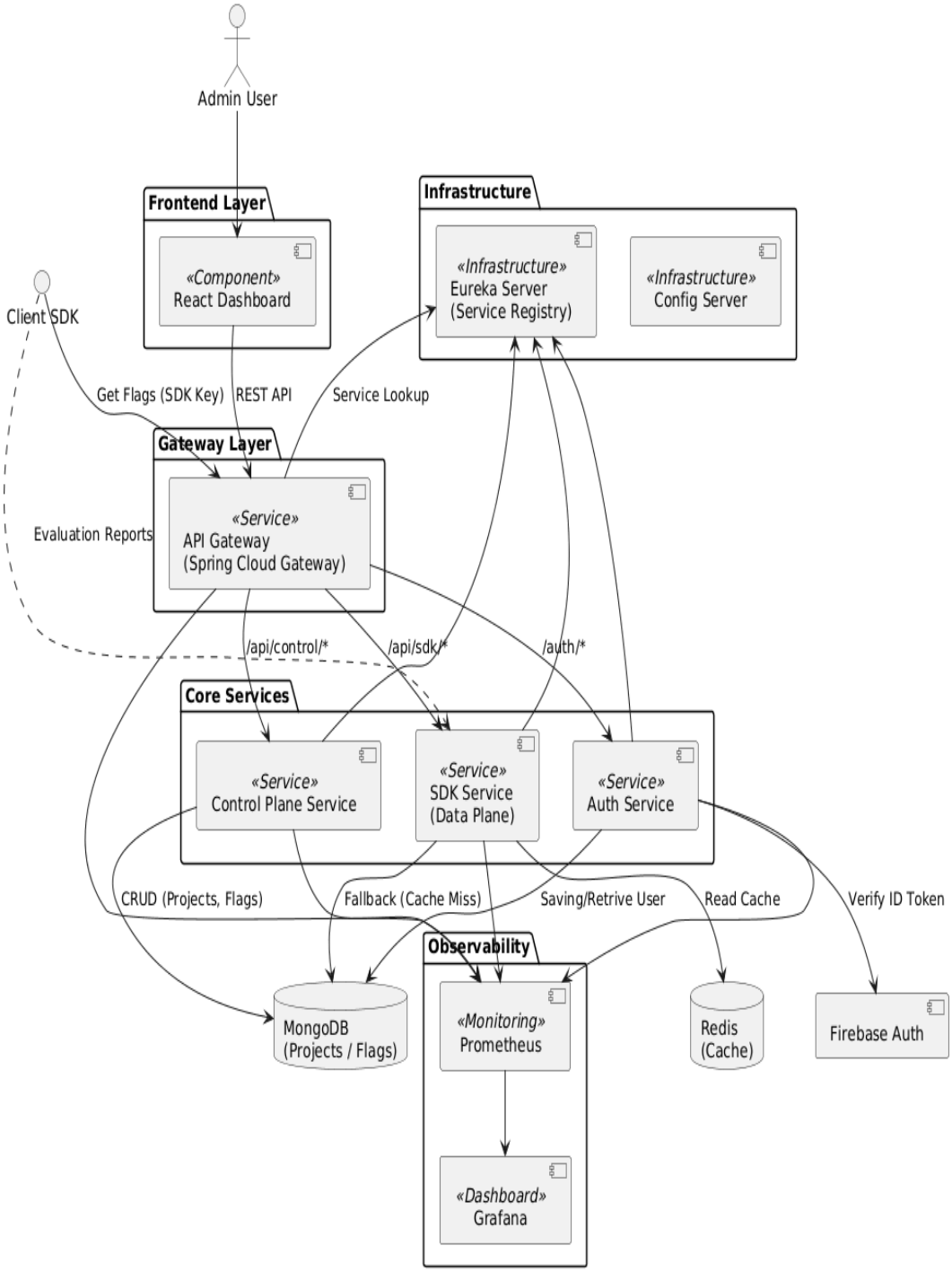

Fig – 5.1: System Design Diagram

At the frontend layer, the system consists of a React-based Dashboard that is used by the Admin User to manage feature flags, projects, and environments. In addition to this, client applications interact with the system using the Client SDK, which is responsible for fetching evaluated feature flags. All incoming requests from both the dashboard and client applications are routed through the API Gateway, which acts as a centralized

entry point for the system. The API Gateway handles request routing, authentication validation, and load distribution across services.

The core services of the system include the Control Plane Service, SDK Service (Data Plane), and Authentication Service. The Control Plane Service is responsible for managing feature flags, including creation, updates, rule configuration, and environment management. The SDK Service is responsible for evaluating feature flags based on targeting rules, dependencies, and rollout percentages, and then delivering the final evaluated configuration to client applications. The Authentication Service ensures secure access by validating JWT tokens and integrating with Firebase Authentication for identity verification.

The infrastructure layer includes the Eureka Server, which enables service discovery and dynamic communication between microservices, and the Config Server, which manages centralized configuration for all services. These components ensure that the system remains flexible and easily configurable across different environments.

For data management, MongoDB is used as the primary database to store projects, environments, and feature flag data, while Redis is used as a caching layer to improve system performance and reduce latency during flag evaluation. Additionally, the system includes an observability layer with tools such as Prometheus for monitoring and Grafana for visualization, which help in tracking system performance and identifying issues in real time.

5.2 MODULE DESIGN

Modules of the System is described below:

- **Authentication Module:** Handles user authentication and authorization using Firebase Authentication and JWT tokens to ensure secure access.
- **API Gateway Module:** Acts as a single-entry point for all requests, performs routing, validation, and security checks.
- **Control Plane Module (Management):** Manages feature flags, projects, and environments, including creation, update, enable/disable, and rule configuration.

- **SDK Service Module (Data Plane):** Evaluates feature flags based on targeting rules, dependencies, and rollout percentage, and returns final values to clients.
- **Service Discovery Module:** Uses Eureka Server to enable dynamic communication between microservices without hardcoded endpoints.
- **Configuration Module:** Managed by Config Server to provide centralized configuration and environment-based settings.
- **Caching Module:** Uses Redis to store frequently accessed data, improving performance and reducing database load.
- **Database Module:** Uses MongoDB to store persistent data such as users, projects, environments, and feature flags.
- **Monitoring Module:** Uses Prometheus and Grafana for system monitoring, metrics collection, and performance visualization.

5.3 DATABASE DESIGN

5.3.1 User Entity

```
public class User {

    @Id
    private String id;

    @Indexed(unique = true)
    @NotBlank(message = "UID is required")
    private String firebaseUid;

    @Indexed(unique = true)
    @NotBlank(message = "Email is required")
    @Email(message = "Invalid email format")
    private String email;

    private String displayName;

    private String pictureUrl;

    private boolean emailVerified;

    private Instant createdAt;

    private Instant updatedAt;

}
```

Fig – 5.2: User Entity

5.3.2 Project Entity

```
public class Project {

    @Id
    private String id;

    private String name;

    private String description;

    private String createdByUid;

    private Instant createdAt;

}
```

Fig – 5.3: Project Entity

5.3.3 Environment Entity

```
public class Environment {

    @Id
    private String id;

    @Indexed
    private String projectId;

    private String name;

    @Indexed(unique = true)
    private String sdkKey;

    private String createdByUid;

    private Instant createdAt;

}
```

Fig – 5.4: Environment Entity

5.3.4 Feature Flag Entity

```java
public class Flag {

    @Id
    private String id;

    @Indexed
    private String environmentId;

    private String key;

    private String displayName;

    private String description;

    private FlagType type;

    private FlagCategory category;

    private Boolean enabled;

    private Object value;

    private Integer rolloutPercentage;

    private List<TargetingRule> targetingRules;

    private RuleNode dependency;

    private Integer version;

    private String createdByUid;

    private Instant createdAt;

    private Instant updatedAt;

}
```

Fig – 5.5: Flag Entity

5.3.5 Targeting Rule Entity

```java
public class TargetingRule {

    private String attribute;

    private TargetOperator operator;

    private Object value;

    private List<Object> values;

}
```

Fig – 5.6: Targeting Rules Entity

5.3.6 Rule Node (Dependency) Entity

```
public class RuleNode {

    private LogicalOperator operator;

    private List<RuleNode> children;

    private DependencyCondition condition;

}
```

Fig – 5.7: Rule Node Entity

5.3.7 Dependency Condition Entity

```
public class DependencyCondition {

    private String flagId;

    private Object expectedValue;

}
```

Fig – 5.8: Dependency Condition Entity

5.3.8 Entity Relationship (ER) Diagram

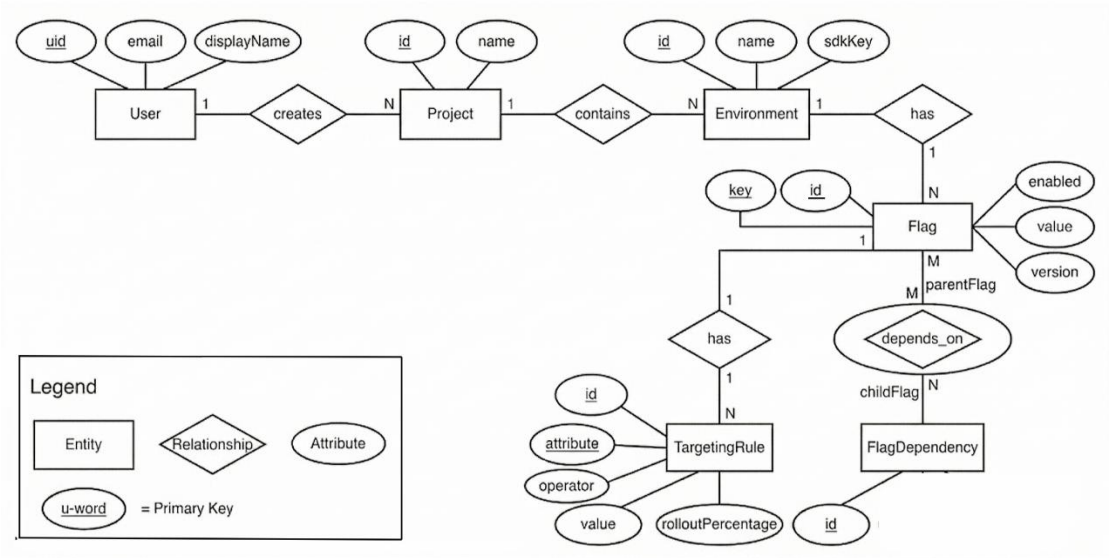

Fig – 5.9: ER Diagram

The Entity Relationship (ER) Diagram represents the overall data structure of the Rollout.io system and defines how different entities are connected with each other. It provides a clear understanding of data organization, relationships, and dependencies among various components of the system.

At the top level, the User entity represents the system users who can create and manage projects. There exists a one-to-many relationship between User and Project, where a single user can create multiple projects, but each project is associated with only one user.

The Project entity acts as a container for organizing environments. A one-to-many relationship exists between Project and Environment, meaning one project can contain multiple environments such as development, staging, and production.

The Environment entity represents different deployment stages and is associated with feature flags. Each environment can have multiple feature flags, establishing a one-to-many relationship between Environment and Feature Flag.

The Feature Flag entity is the core entity of the system. It stores configuration data such as key, value, enabled status, and version. Each feature flag belongs to one environment but can have multiple associated rules and dependencies.

The Targeting Rule entity is connected to the Feature Flag entity in a one-to-many relationship. This means a single feature flag can have multiple targeting rules, which define conditions based on user attributes for feature activation.

The Flag Dependency entity represents relationships between feature flags. It defines how one feature flag depends on another using parent-child relationships. This enables complex feature control mechanisms where a feature is activated only if certain dependent conditions are satisfied.

Additionally, the system supports hierarchical dependency logic using rule nodes, allowing multiple conditions to be combined using logical operators such as AND and OR.

5.3.9 Class Diagram

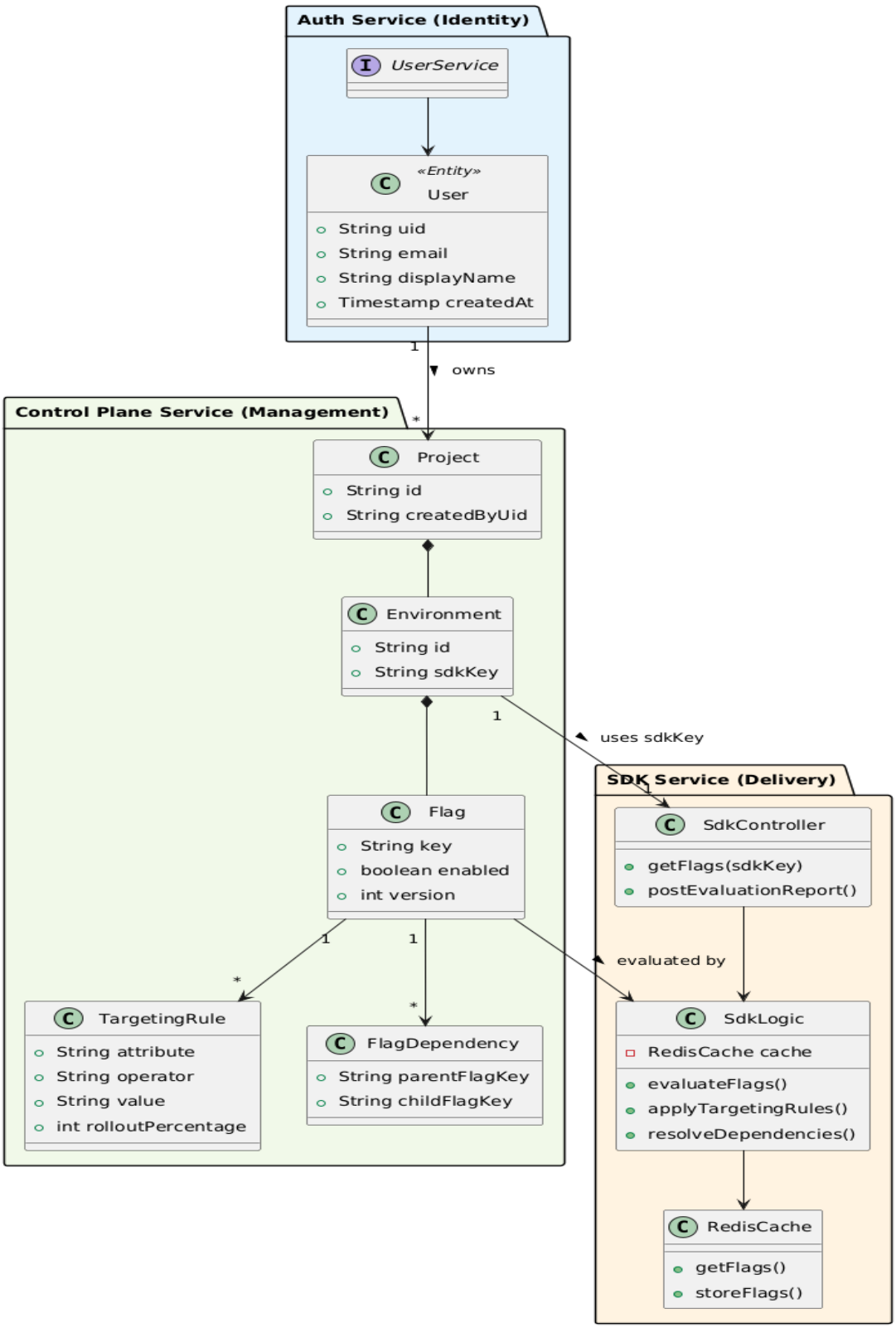

Fig – 5.10: Class Diagram

The Class Diagram represents the object-oriented structure of the Rollout.io system and illustrates how different classes interact across various services. It provides a detailed view of entities, service components, and their relationships, reflecting the internal design of the system.

The diagram is divided into three main layers: the Auth Service (Identity), the Control Plane Service (Management), and the SDK Service (Delivery). The Auth Service manages user identity through the User entity, which is responsible for storing user-related information. A one-to-many relationship exists between User and Project, indicating that a single user can own multiple projects.

Within the Control Plane Service, the Project class acts as a container for multiple environments. Each Environment is associated with a unique SDK key and contains multiple Feature Flags. The Flag class is central to the system and includes attributes such as key, enabled status, and version. Each flag can have multiple Targeting Rules, which define conditions based on user attributes, and multiple Flag Dependencies, which define relationships between flags.

The SDK Service is responsible for evaluating and delivering feature flags. The SDK Controller class exposes endpoints for retrieving flags and sending evaluation reports. The SDK Logic class performs core evaluation operations such as applying targeting rules and resolving dependencies. It interacts with the Redis Cache class to fetch and store flag data efficiently, ensuring low-latency responses.

5.4 API DESIGN

The API Design of the Rollout.io system defines how different components communicate with each other using RESTful principles. All requests from the frontend dashboard and client applications are routed through the API Gateway, which handles request validation, authentication, and routing to appropriate microservices.

The APIs are divided into three main categories: Authentication APIs, Feature Flag APIs, and SDK APIs. These APIs enable secure user access, management of feature flags, and delivery of evaluated configurations to client applications, ensuring efficient and scalable system operation.

5.4.1 Auth Service APIs (User Management)

Fig – 5.11: User Management APIs

5.4.2 Control Plane Service APIs (Project Management)

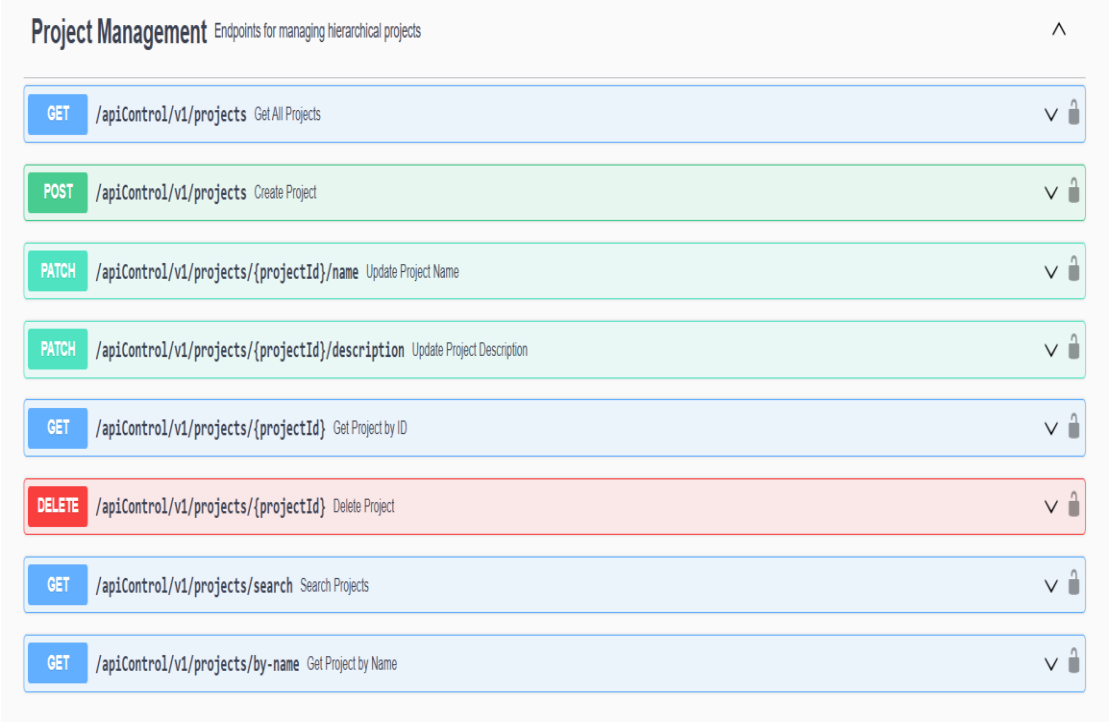

Fig – 5.12: Project Management APIs

5.4.3 Control Plane Service APIs (Environment Management)

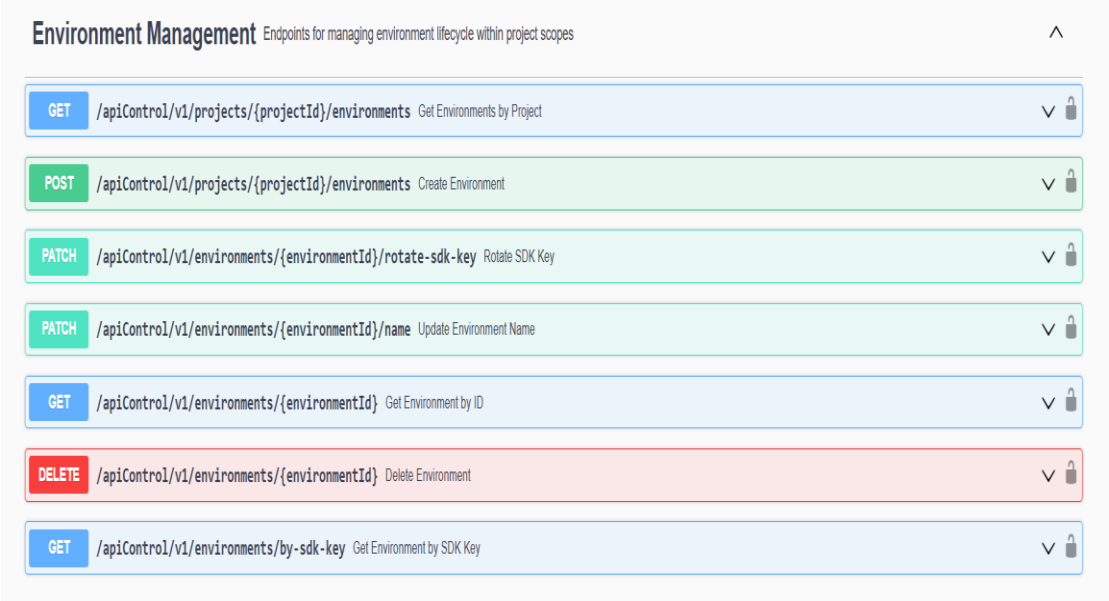

Fig – 5.13: Environment Management APIs

5.4.4 Control Plane Service APIs (Core Flag Management)

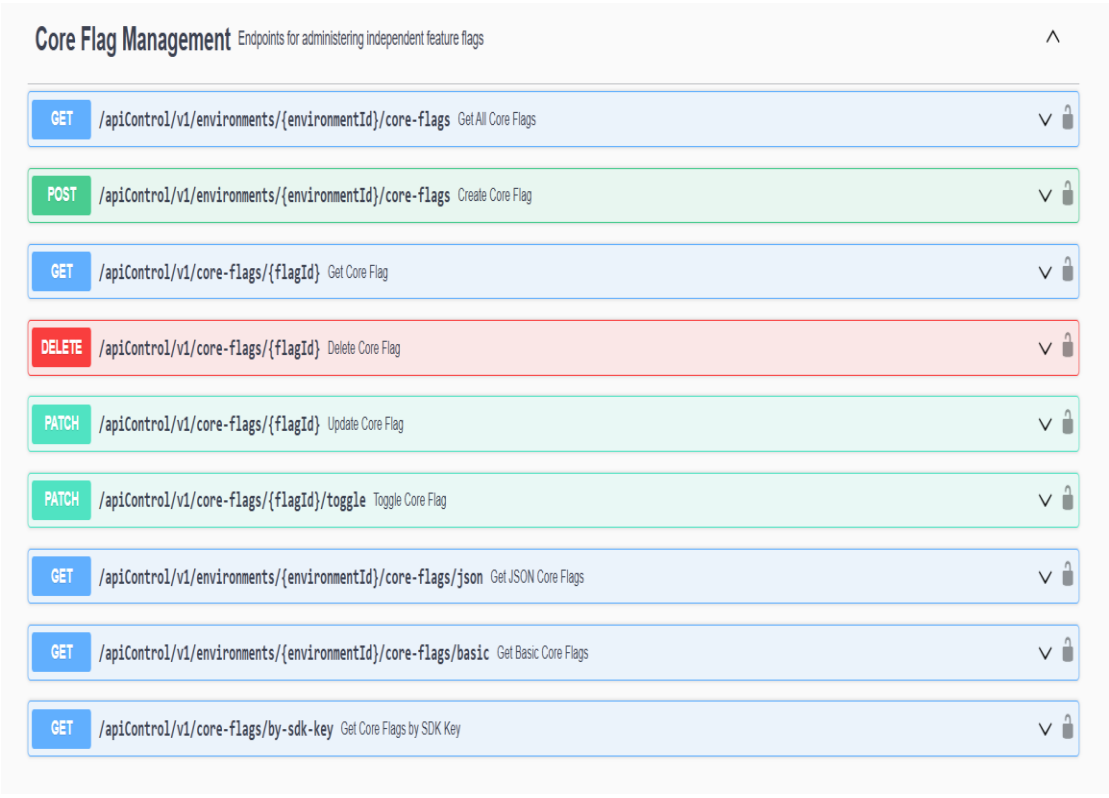

Fig – 5.14: Core Flag Management APIs

5.4.5 Control Plane Service APIs (Dependent Flag Management)

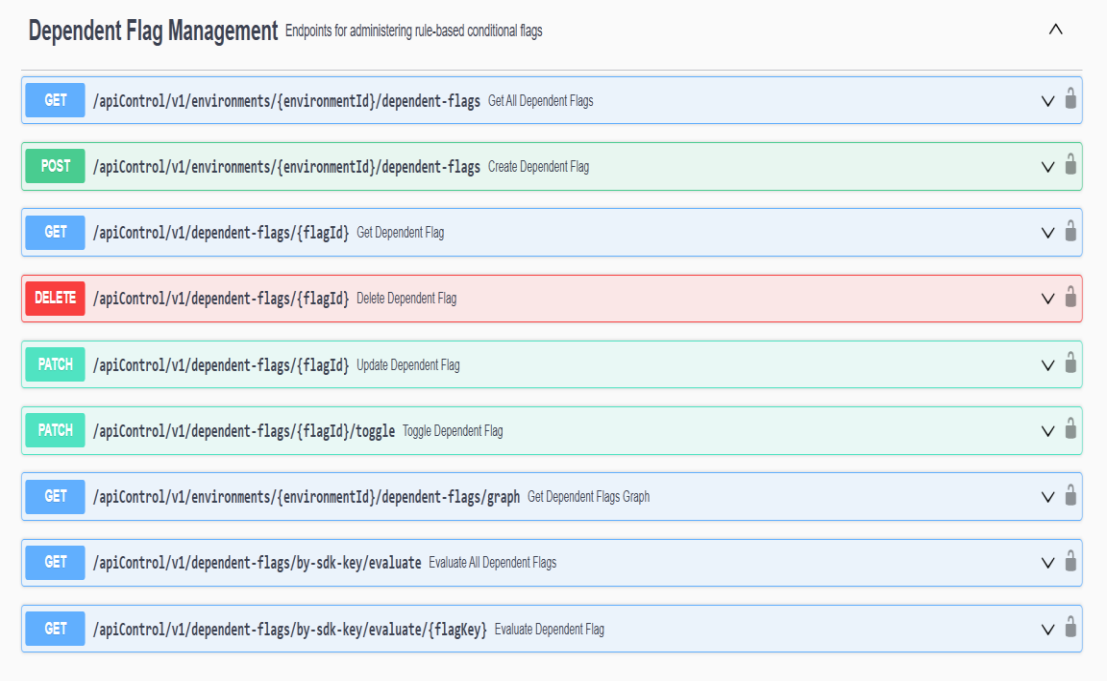

Fig – 5.15: Dependent Flag Management APIs

5.4.6 Control Plane Service APIs (Audit Logs)

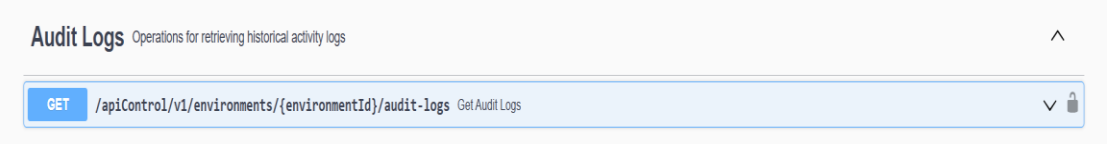

Fig – 5.16: Audit Logs APIs

5.4.7 SDK Service APIs (Admin Management)

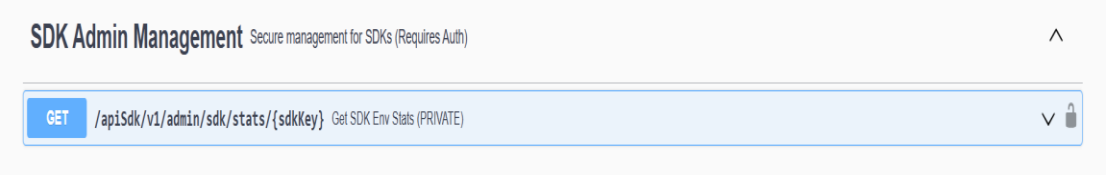

Fig – 5.17: SDK Admin APIs

5.4.8 SDK Service APIs (Public Access)

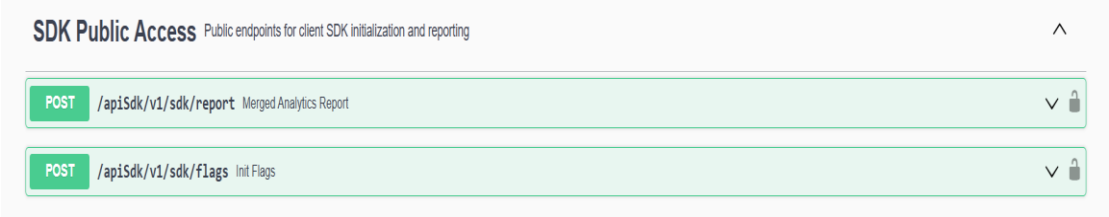

Fig – 5.18: SDK Public APIs

5.5 DEPLOYMENT ARCHITECTURE

5.5.1 Deployment Diagram

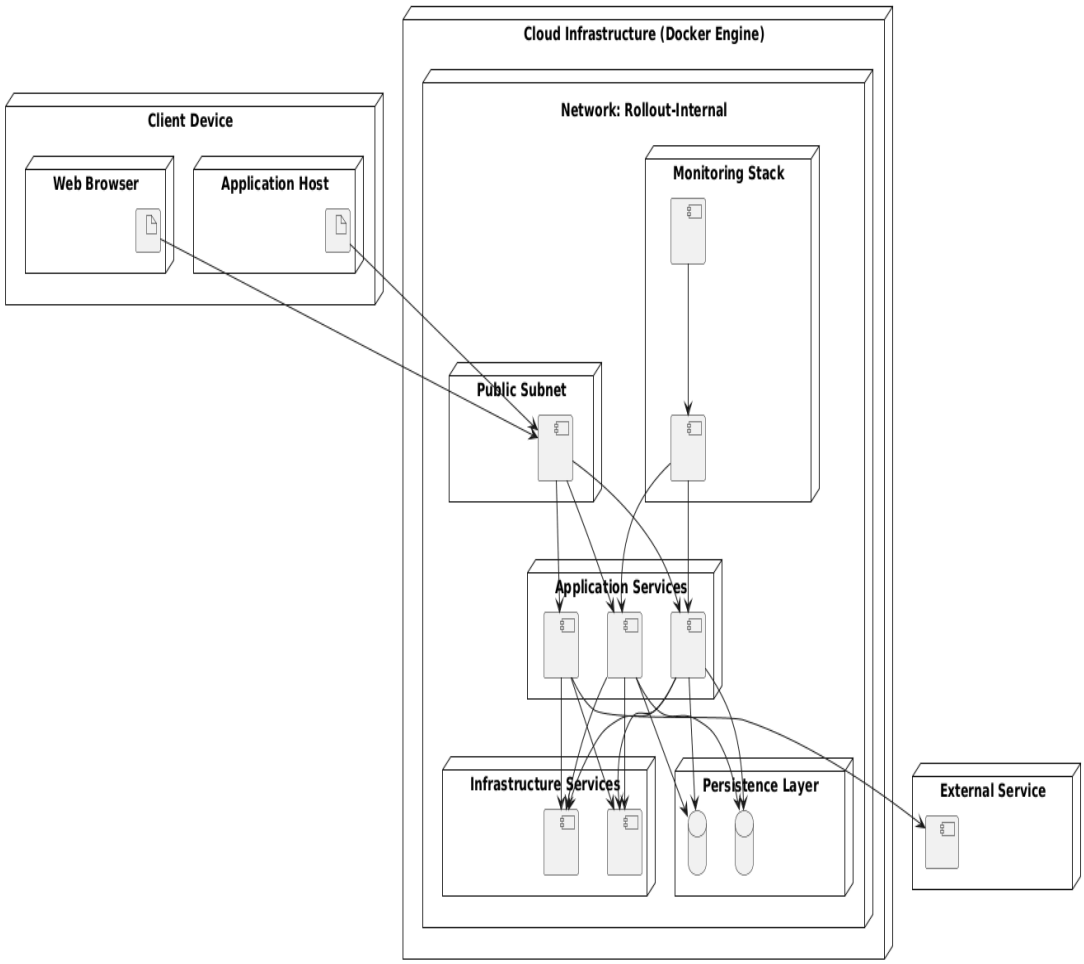

Fig – 5.19: Deployment Diagram

The Deployment Diagram illustrates the physical architecture of the Rollout.io system and shows how different components are deployed across the infrastructure. The system is hosted within a cloud environment using Docker-based containerization, ensuring scalability and isolation of services.

On the client side, users interact with the system through a web browser or application host. These clients communicate with the backend through the public subnet, where incoming requests are routed into the system.

Within the cloud infrastructure, all services are deployed inside a private internal network (Rollout-Internal). The Application Services layer consists of core microservices such as the API Gateway, Control Plane Service, SDK Service, and Auth Service. These services communicate with each other internally to process requests and deliver responses.

The Infrastructure Services layer includes essential components such as service discovery and configuration management, enabling smooth communication between microservices. The Persistence Layer contains databases like MongoDB for storing application data and Redis for caching frequently accessed data to improve performance.

The system also integrates a Monitoring Stack, including tools like Prometheus and Grafana, to track system metrics and performance. Additionally, external services such as Firebase Authentication are used for secure user authentication.

5.6 TECHNOLOGY STACK JUSTIFICATION

The Rollout.io system is built using a modern and scalable technology stack that supports microservices architecture, high performance, and flexibility. Each technology has been selected based on its suitability for handling distributed systems, real-time configuration delivery, and ease of development.

Backend – Spring Boot: Spring Boot is used for developing backend microservices due to its robustness, scalability, and strong ecosystem. It simplifies the development of REST APIs and supports features like dependency injection, security integration, and

easy configuration management. It is well-suited for building enterprise-level distributed systems.

Frontend – React: React is used for building the frontend dashboard because of its component-based architecture and efficient state management. It enables the development of dynamic and responsive user interfaces, making it easier to manage feature flags, projects, and environments.

Database – MongoDB: MongoDB is used as the primary database because of its flexible schema design. Since feature flags and targeting rules can have dynamic and nested structures, a NoSQL database like MongoDB is more suitable than traditional relational databases.

Caching – Redis: Redis is used as a caching layer to improve performance and reduce database load. Frequently accessed data such as feature flags are stored in Redis, allowing faster retrieval during runtime evaluation.

API Gateway – Spring Cloud Gateway: Spring Cloud Gateway is used to route incoming requests to appropriate microservices. It also handles cross-cutting concerns such as authentication, logging, and request filtering, improving system security and maintainability.

Service Discovery – Eureka: Eureka is used for service discovery, allowing microservices to dynamically register and discover each other. This helps in maintaining loose coupling between services and supports scalability.

Authentication – Firebase Authentication: Firebase Authentication is used for secure user authentication and identity management. It provides JWT-based authentication, reducing the need to build a custom authentication system and ensuring high security standards.

Containerization – Docker: Docker is used to containerize all services, ensuring consistency across development and deployment environments. It simplifies deployment and enables easy scaling of services.

Monitoring – Prometheus and Grafana: Prometheus is used for collecting system metrics, while Grafana is used for visualizing these metrics. Together, they provide observability into system performance, helping in monitoring and debugging.

5.7 SECURITY DESIGN

Authentication using Firebase JWT: User authentication is handled using Firebase Authentication. After successful login, a JWT (JSON Web Token) is issued, which is included in every request from the frontend. This token is validated by the API Gateway and backend services to verify the identity of the user.

API Gateway Token Validation: All incoming requests first pass through the API Gateway, which acts as a centralized security layer. The gateway validates the JWT token using Firebase public keys and only forwards authenticated requests to internal microservices. This prevents unauthorized access to backend services.

User-Based Access Control: The system follows a user-based access model where all resources such as projects, environments, and feature flags are linked to a specific user (createdByUid). Each request is processed in the context of the authenticated user, ensuring that users can only access and modify their own resources.

Secure SDK Access using SDK Keys: Client applications do not use JWT tokens. Instead, they access the system using environment-specific SDK keys. These keys are validated by the SDK Service before serving feature flags. This ensures that only authorized client applications can fetch configuration data.

Internal Service Security: All backend services are deployed within a private internal network. Direct access to services like Control Plane and SDK Service is restricted, and communication happens only through the API Gateway. This reduces exposure to external threats.

CHAPTER 6: IMPLEMENTATION

6.1 INTRODUCTION

The implementation of the Rollout.io system focuses on developing a complete end-to-end feature flag management platform capable of controlling application behaviour dynamically without requiring redeployment. The system is designed using a microservices-based architecture, ensuring scalability, flexibility, and maintainability across different components.

The backend of the system is implemented using Spring Boot and is divided into multiple microservices, including the Authentication Service, Control Plane Service, and SDK Service. The Authentication Service handles user identity using Firebase Authentication and JWT tokens. The Control Plane Service manages core functionalities such as project creation, environment management, feature flag configuration, targeting rules, and dependency handling. The SDK Service is responsible for evaluating feature flags in real-time and delivering the final configuration to client applications efficiently using caching mechanisms like Redis.

The frontend of the system is developed using React, providing an interactive dashboard for managing projects, environments, and feature flags. The dashboard allows users to create and control feature flags, define targeting rules, manage dependencies, and monitor system behaviour through a user-friendly interface.

A custom SDK has also been implemented to enable seamless integration of feature flags into client applications. The SDK communicates with the backend services using SDK keys, fetches feature configurations, and applies evaluation logic based on user context, targeting rules, and dependency conditions.

To demonstrate the practical application of the system, a demo web application inspired by a Zomato-like interface was developed. This application integrates the SDK and uses feature flags to dynamically control UI elements such as feature visibility, promotional banners, and functionality toggles. This real-world simulation highlights the effectiveness of the system in managing features in a live environment.

6.2 BACKEND IMPLEMENTATION

6.2.1 Microservices Architecture Implementation

The backend of the Rollout.io system is implemented using a microservices-based architecture, where the application is divided into multiple independent services. Each service is developed as a separate Spring Boot application and is responsible for handling a specific part of the system. This modular design improves scalability, maintainability, and fault isolation.

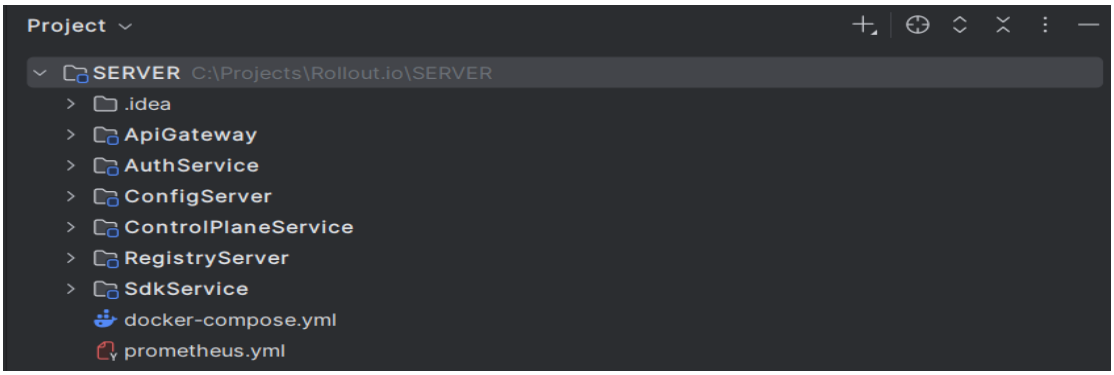

Fig – 6.1: Project Structure/ Microservices Implementation

Auth Service: The Auth Service manages user authentication and identity. It integrates with Firebase Authentication and validates JWT tokens received from the client. It also provides APIs for managing user profiles such as fetching user details, updating profile information, and deleting accounts. This service ensures that all user-related operations are secure and authenticated.

Control Plane Service: The Control Plane Service is the core management service of the system. It handles all administrative operations related to feature flag management. This includes:

- Project creation and management
- Environment configuration
- Feature flag creation and updates
- Targeting rule definition
- Dependency management between flags

- Audit logging of system activities

This service interacts with the database (MongoDB) to store and manage all configuration data.

SDK Service: The SDK Service acts as the data plane and is responsible for real-time evaluation and delivery of feature flags to client applications. It processes incoming requests from SDK clients using SDK keys and performs the following:

- Fetches flag configurations
- Applies targeting rules
- Resolves dependencies between flags
- Returns evaluated flag values

To improve performance, it uses Redis as a caching layer to reduce database load and ensure low-latency responses.

Config Server: The Config Server is used to manage centralized configuration for all microservices. It allows services to fetch their configuration dynamically, making it easier to manage environment-specific properties without hardcoding them into each service.

Registry Server (Eureka): The Registry Server is used for service discovery. All microservices register themselves with the Eureka server, allowing other services to discover and communicate with them dynamically. This eliminates the need for hardcoded service URLs and supports scalability.

Monitoring (Prometheus): The system includes a monitoring setup using Prometheus, which collects metrics from different services. These metrics help in tracking system performance, detecting issues, and analysing request patterns.

Containerization (Docker Compose): All services are containerized using Docker and orchestrated using Docker Compose. This ensures that the entire system can be deployed consistently across different environments. Each service runs in its own container, enabling easy scaling and isolation.

6.2.2 REST API Implementation

The APIs follow standard HTTP methods such as GET, POST, PATCH, and DELETE for performing operations like retrieving data, creating resources, updating configurations, and deleting entities. All APIs are structured using consistent URL patterns and versioning (e.g., /apiControl/v1/..., /apiAuth/v1/..., /apiSdk/v1/...) to maintain clarity and scalability.

Authentication is handled using JWT tokens for protected endpoints. The API Gateway validates incoming requests before forwarding them to respective services. For SDK-based public APIs, access is controlled using SDK keys instead of user authentication.

Controller Layer: The controller layer is responsible for handling incoming HTTP requests and mapping them to appropriate service methods. Each controller corresponds to a specific domain of the system:

- ProjectController → Project management
- EnvironmentController → Environment operations
- CoreFlagController → Feature flag management
- DependentFlagController → Dependency-based flags
- AuditLogController → Audit log retrieval

These controllers define REST endpoints using annotations such as @GetMapping, @PostMapping, @PatchMapping, and @DeleteMapping.

Service and Logic Layer: The service layer acts as an intermediary between controllers and business logic. It ensures proper processing of requests and delegates complex operations to dedicated logic classes.

The logic layer contains the core implementation of system behavior, including:

- Feature flag evaluation
- Targeting rule processing
- Dependency resolution
- Environment and project operations

Examples include:

- CoreFlagServiceLogic

- DependentFlagServiceLogic
- ProjectServiceLogic

This separation allows better organization of business logic and makes the system easier to extend.

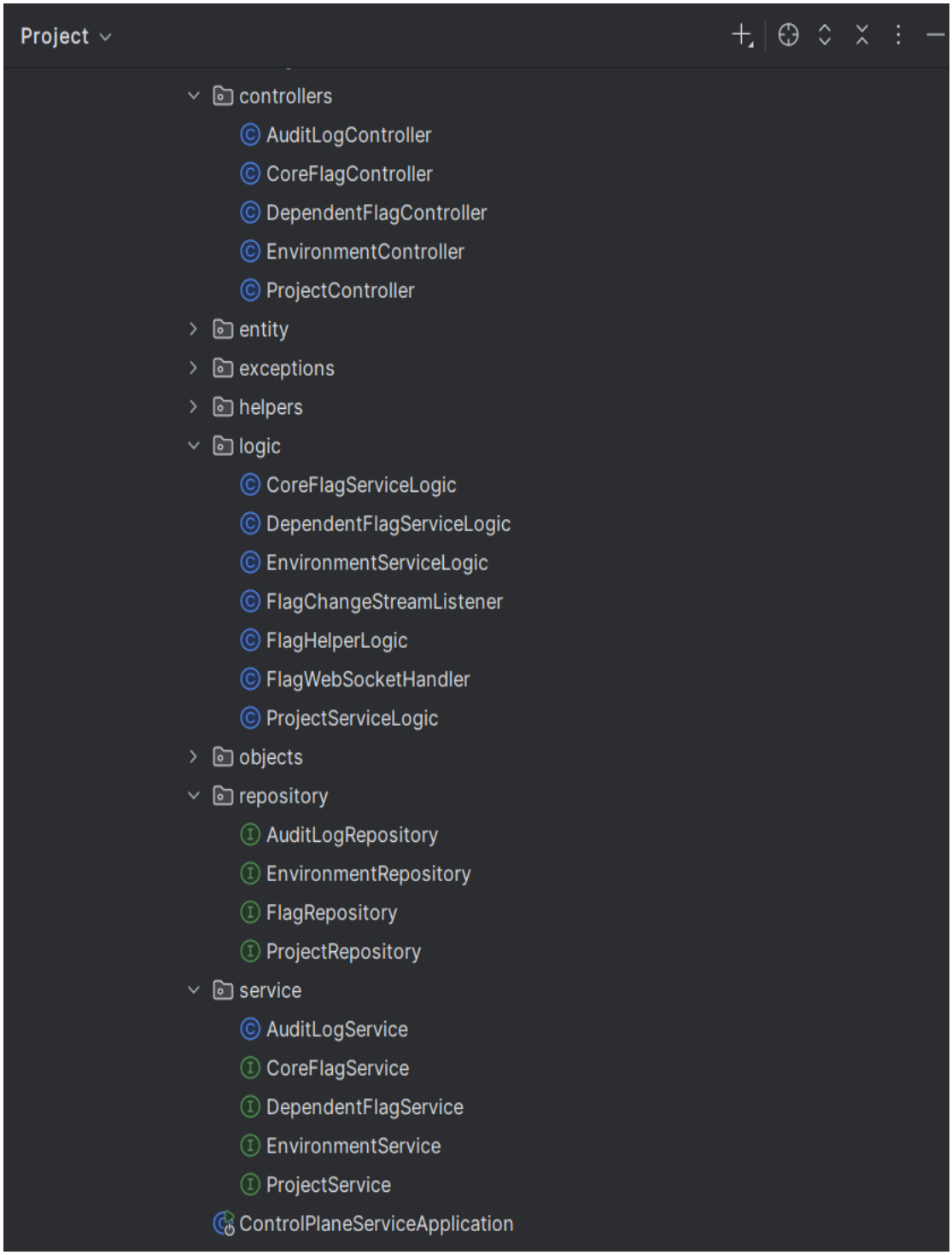

Fig – 6.2: REST APIs implementation classes

Repository Layer: The repository layer is responsible for interacting with the database. It uses Spring Data MongoDB to perform CRUD operations. These repositories provide an abstraction over database queries, reducing the need for manual query handling.

Entity and Object Layer: The entity layer defines the data models used for storing information in the database, such as User, Project, Environment, and Flag. Additionally, object classes (DTOs) are used for transferring data between layers in a structured manner.

Exception and Helper Handling: The system includes dedicated packages for exception handling and helper utilities. This ensures consistent error responses and reusable utility functions across the application.

6.2.3 Database Implementation

The Rollout.io system uses MongoDB as the primary database for storing all application data. MongoDB is a NoSQL database that provides a flexible and schema-less structure, making it suitable for handling dynamic data such as feature flags, targeting rules, and dependency configurations.

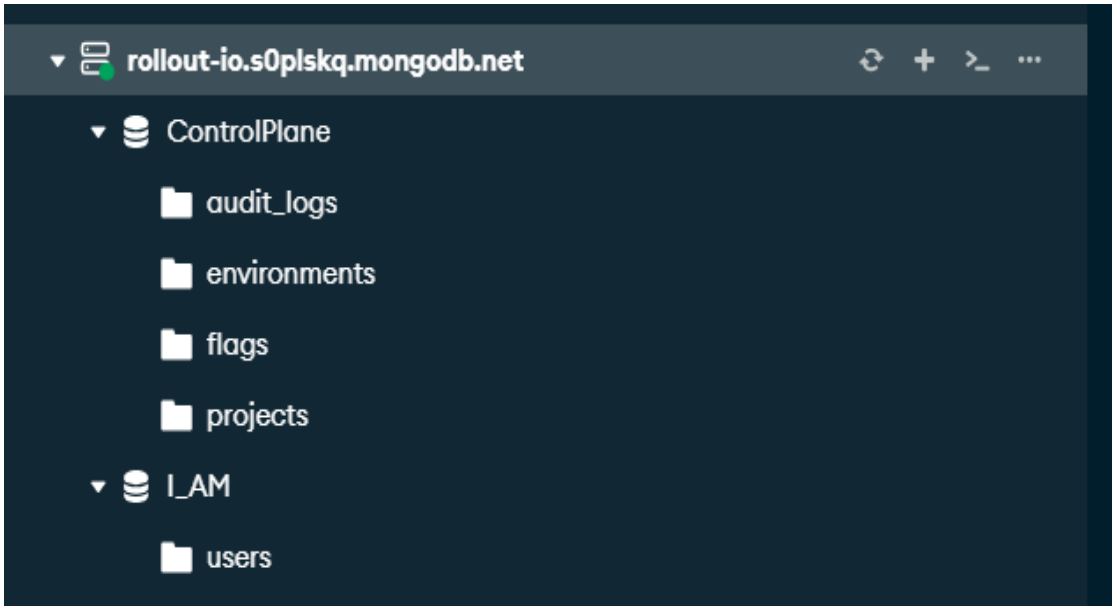

Fig – 6.3: MongoDB Compass Databases

Database Structure

The system organizes data into multiple collections grouped under different databases based on service responsibilities.

- **ControlPlane Database**
  - projects: Stores project details
  - environments: Stores environment configurations with SDK keys
  - flags: Stores feature flag configurations
  - audit_logs: Stores system activity logs
- **I_AM Database**
  - users → Stores user identity and profile information

This separation ensures clear responsibility between identity management and feature configuration management.

Data Relationships

MongoDB is a NoSQL database, logical relationships are maintained using identifiers:

- A **User** creates multiple **Projects**
- A **Project** contains multiple **Environments**
- An **Environment** contains multiple **Feature Flags**
- A **Flag** can include targeting rules and dependency conditions

These relationships are maintained using fields such as projectId, environmentId, and createdByUid.

Handling Dynamic Data

Feature flags may contain nested structures such as targeting rules and dependency trees. MongoDB allows storing these structures directly within documents, eliminating the need for complex joins and improving query performance.

Database Access Layer

The system uses Spring Data MongoDB repositories to perform database operations. These repositories provide built-in support for CRUD operations and simplify interaction with the database.

6.2.4 Caching Implementation

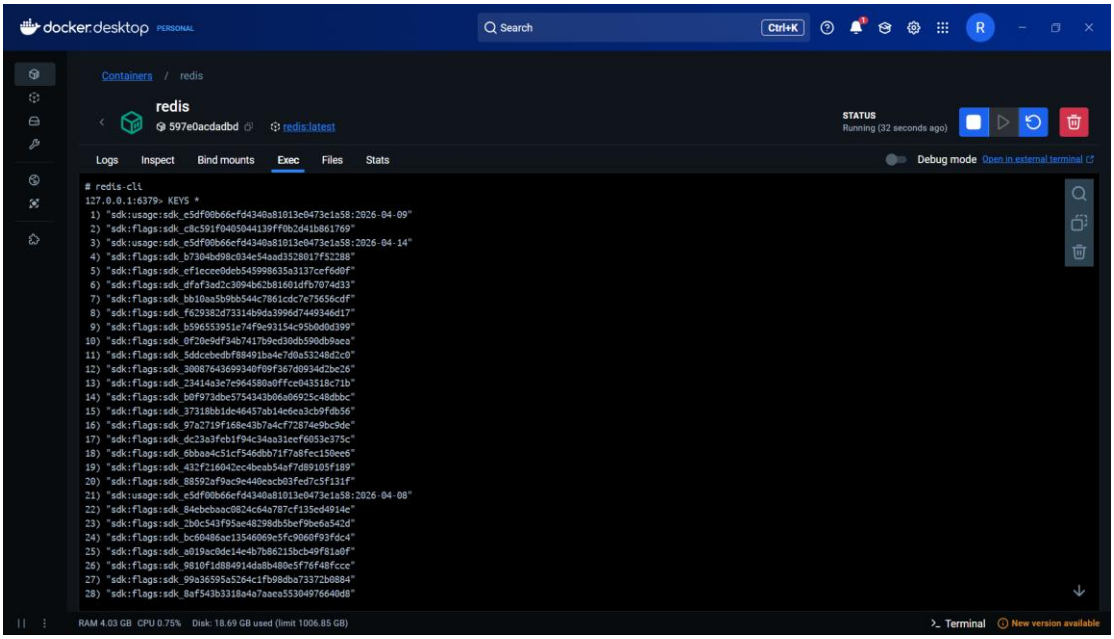

Fig – 6.4: Redis Cached data keys

Purpose of Caching

Caching is implemented to:

- Reduce latency during SDK flag fetching
- Avoid repeated database queries for frequently accessed flags
- Improve performance of real-time feature evaluation
- Efficiently track and store usage analytics

SDK Flag Caching (TTL-Based)

When client application requests feature flags using the SDK, the evaluated flag response is cached in Redis for a short duration.

- The cache is stored using SDK key-based identifiers
- Each cached entry has a **Time-To-Live (TTL) of 30 seconds**
- During this time, repeated requests for the same SDK key are served directly from Redis

This approach ensures that:

- The system avoids redundant computations

- Response time is significantly reduced
- Updated flag configurations are reflected after a short interval

Usage Reporting Caching

The system also uses Redis to temporarily store feature usage data reported by client applications.

- Usage events are aggregated in Redis
- Keys follow a structured format such as: sdk:usage:{sdkKey}:{date}
- This reduces direct writes to the database
- Improves performance for high-frequency reporting

Caching Workflow

The caching process follows these steps:

1. SDK request is received
2. Redis is checked for cached flag response
3. If available, cached data is returned (**cache hit**)
4. If not available, flags are fetched from MongoDB and evaluated
5. The result is stored in Redis with a TTL of 30 seconds
6. Usage data is recorded and temporarily stored in Redis

Cache Key Structure

The system uses structured keys for efficient access:

- sdk:flags:{sdkKey} → Cached evaluated flags (TTL: 30 seconds)
- sdk:usage:{sdkKey}:{date} → Usage analytics data

6.2.5 API Documentation using Swagger

The Rollout.io system uses Swagger (OpenAPI) for automatic API documentation and interactive testing of all backend services. Swagger provides a centralized interface where all REST APIs across different microservices can be explored, tested, and validated.

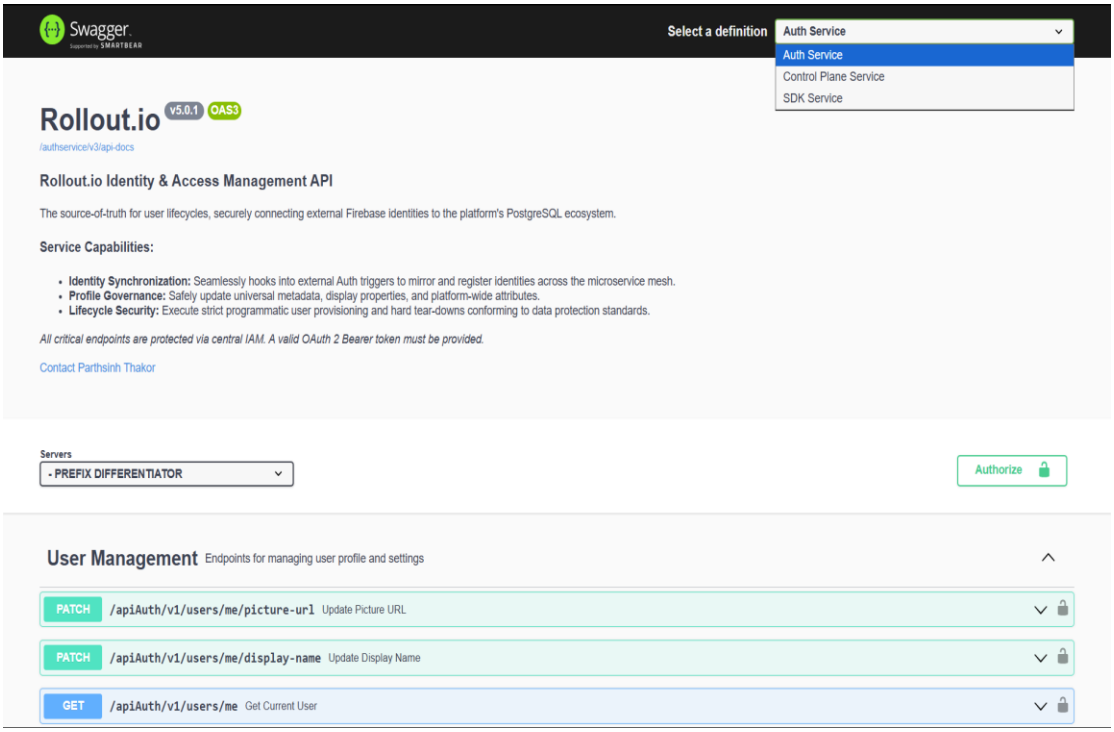

Fig – 6.5: API Documentation

Purpose of Swagger Integration

Swagger is integrated to:

- Provide a clear and structured view of all available APIs
- Enable direct API testing without external tools
- Improve developer understanding of request and response formats
- Simplify debugging and development

Implementation

Swagger is implemented using the Springdoc OpenAPI library. Each microservice defines its API documentation using annotations such as:

- @Operation → Describes endpoint functionality
- @Tag → Groups related APIs
- @RequestParam, @PathVariable → Define inputs

All service-level API documentations are aggregated and accessible through a unified Swagger UI interface.

Multi-Service API Documentation

The system supports multiple microservices, and Swagger allows switching between them using a dropdown selector. The available services include:

- Auth Service
- Control Plane Service
- SDK Service

This enables developers to explore APIs specific to service within a single interface.

Features of Swagger UI

The Swagger interface provides:

- Categorized API endpoints (e.g., User Management, Project Management)
- HTTP methods such as GET, POST, PATCH, DELETE
- Input fields for parameters and request bodies
- "Authorize" option for secured endpoints using JWT tokens
- Real-time API execution and response visualization

6.2.6 API Execution and Response Handling

The backend APIs of the Rollout.io system are designed to process client requests and return structured responses that can be directly consumed by frontend applications and SDK clients. These APIs handle operations such as feature flag evaluation, project management, and environment configuration.

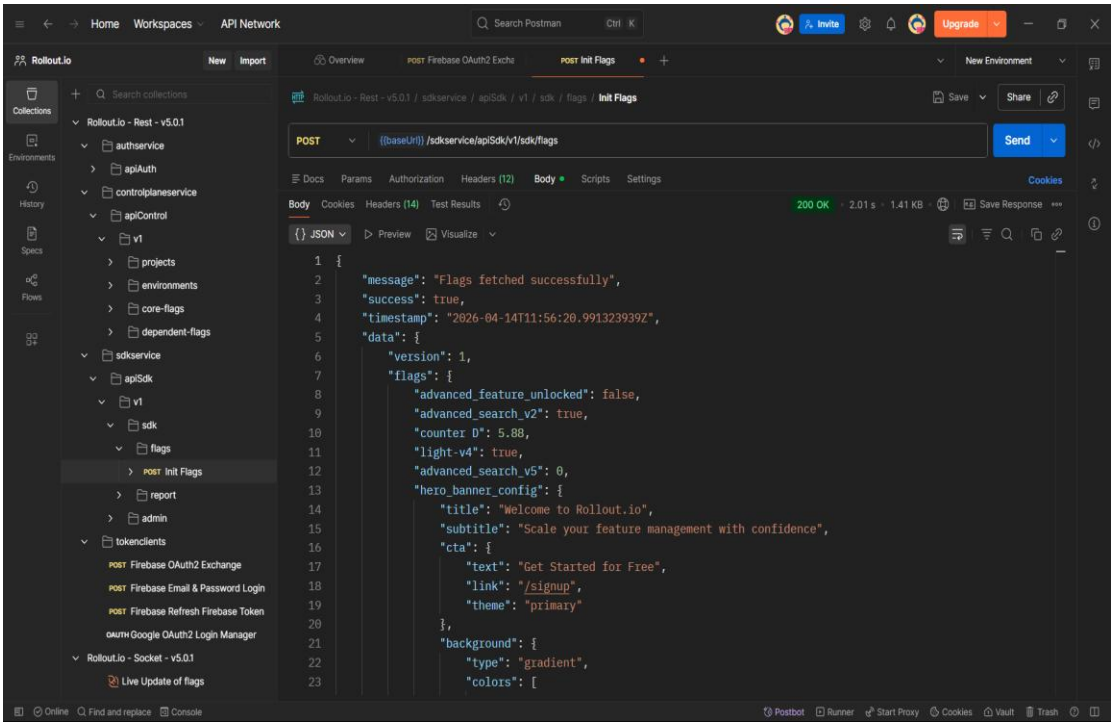

Fig – 6.6: SDK API Response Showing Evaluated Feature Flags

Request Processing Flow

When a request is received:

1. The request is routed through the API Gateway
2. The appropriate controller handles the request
3. The request is processed by the service and logic layers
4. Business logic such as feature flag evaluation is executed
5. A structured response is generated and returned to the client

Standard Response Structure

All APIs follow a consistent response format to ensure easy integration across different components:

- **success / status** → Indicates the result of the operation
- **message** → Provides a descriptive response message
- **data** → Contains the actual payload returned by the system

Example API Response


```
{
  "message": "Flags fetched successfully",
  "success": true,
  "timestamp": "2026-04-14T11:56:20.991Z",
  "data": {
    "version": 1,
    "flags": {
      "advanced_feature_unlocked": false,
      "advanced_search_v2": true,
      "light-v4": true
    }
  }
}
```


Fig – 6.7: Example API Response

6.2.7 Monitoring and Visualization using Grafana

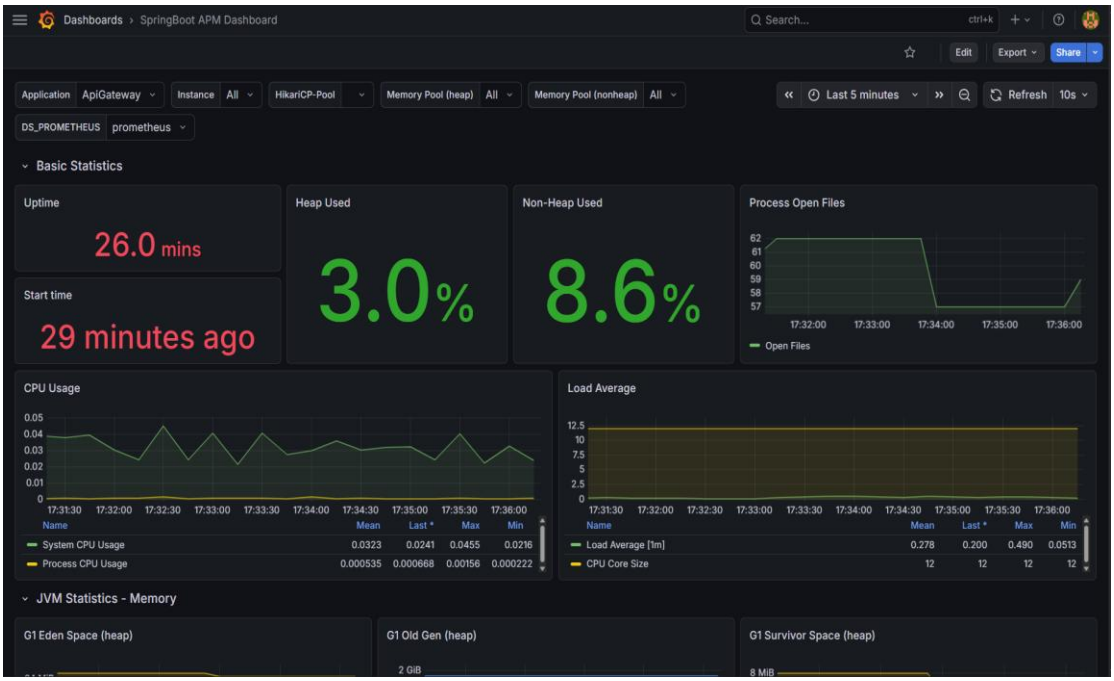

Fig – 6.8: Grafana Dashboard

Purpose of Monitoring

Monitoring is implemented to:

- Track system performance and uptime
- Analyze CPU and memory usage

- Monitor API request handling
- Detect performance bottlenecks

Prometheus Integration

Prometheus is used as the metrics collection system. It periodically collects metrics from various microservices such as:

- API Gateway
- Auth Service
- Control Plane Service
- SDK Service

These metrics include system-level and application-level data such as request count, latency, and resource usage.

Grafana Dashboard

Grafana is used to visualize the collected metrics in the form of interactive dashboards. The dashboard provides real-time insights into system performance and resource utilization.

The implemented dashboard includes:

- **Uptime Monitoring** → Tracks how long the service has been running
- **Memory Usage (Heap and Non-Heap)** → Displays JVM memory consumption
- **CPU Usage** → Shows system and process-level CPU utilization
- **Load Average** → Indicates system load over time
- **Process Open Files** → Tracks system resource usage

6.2.8 Containerization using Docker

The Rollout.io system is fully containerized using Docker to ensure consistent deployment, scalability, and efficient management of all microservices. Each service is packaged as an independent container, allowing the system to run reliably across different environments.

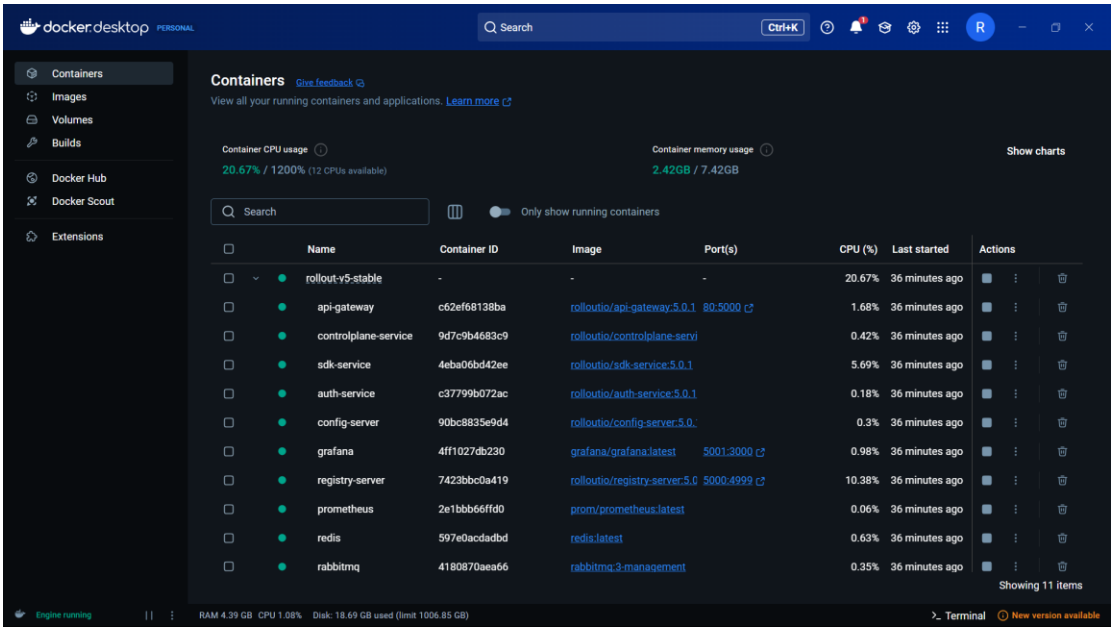

Fig – 6.9: Docker Compose Container Local Deployment

Purpose of Containerization

Docker is used in the system to:

- Standardize deployment across environments
- Isolate microservices for independent execution
- Simplify setup and dependency management
- Enable scalability and easier maintenance

Docker Compose Architecture

The system uses Docker Compose to orchestrate all services within a unified environment. All containers are connected through a shared internal network, enabling seamless communication between services.

The deployed containers include:

- API Gateway
- Auth Service
- Control Plane Service
- SDK Service
- Config Server
- Registry Server (Eureka)
- Redis (Caching Layer)

- Prometheus (Monitoring)
- Grafana (Visualization)
- RabbitMQ (Messaging Queue)

Service Execution

Each service runs in its own isolated container, ensuring:

- Independent lifecycle management
- Fault isolation
- Efficient resource utilization

The API Gateway acts as the entry point, while other services communicate internally through service discovery and REST APIs.

6.2.9 SDK Integration and Usage

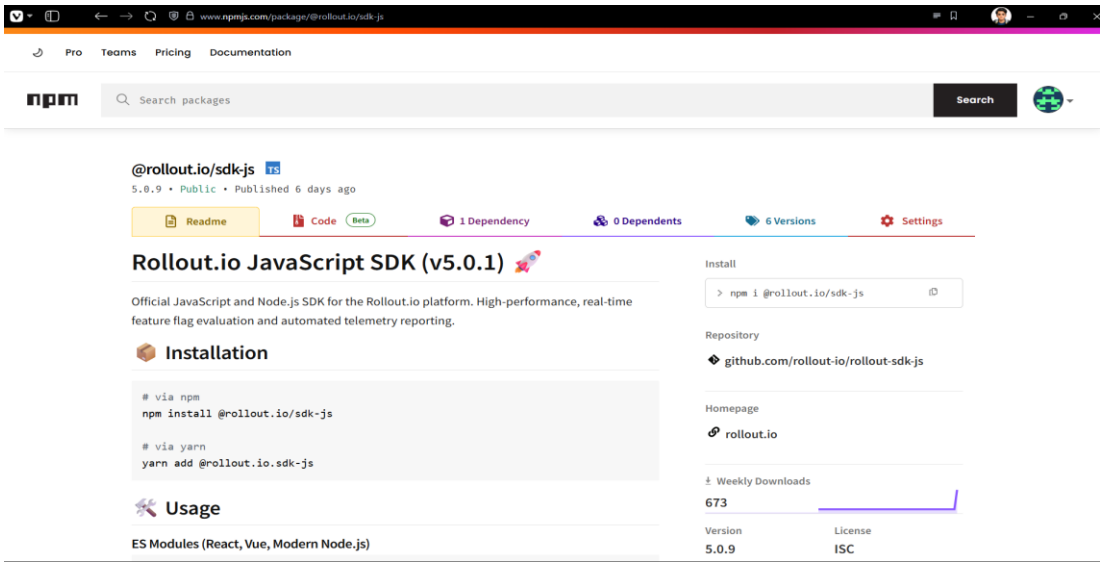

Fig – 6.10: Rollout SDK NPM package

Purpose of SDK

The SDK is designed to:

- Fetch evaluated feature flags from the backend
- Enable real-time feature toggling in applications
- Provide a simple integration interface for developers
- Support telemetry reporting and analytics

NPM-Based Distribution

The SDK is published as an npm package, making it easily accessible and installable in JavaScript-based applications.

npm install @rollout.io/sdk-js

This allows developers to quickly integrate the SDK into modern frameworks such as React, Vue, and Node.js applications.

SDK Workflow

SDK workflow includes:

1. The client initializes the SDK using a unique SDK key
2. The SDK sends a request to the backend (/apiSdk/v1/sdk/flags)
3. The backend evaluates feature flags based on rules and dependencies
4. The evaluated flags are returned to the SDK
5. The SDK applies these flags within the application

Usage in Client Applications

The SDK is integrated into frontend applications such as dashboards or websites. It dynamically controls feature availability without requiring application redeployment, enabling real-time configuration updates.

6.3 FRONTEND IMPLEMENTATION

The Rollout.io system includes a frontend interface that allows users to manage feature flags through an interactive dashboard. This dashboard communicates directly with backend services using REST APIs for operations such as project creation, environment management, and flag configuration.

6.3.1 Landing Page (Launch Website)

The landing page is designed with a modern UI and visually appealing layout to clearly communicate the platform's purpose. It emphasizes real-time configuration, low-latency execution, and support for distributed microservices architectures.

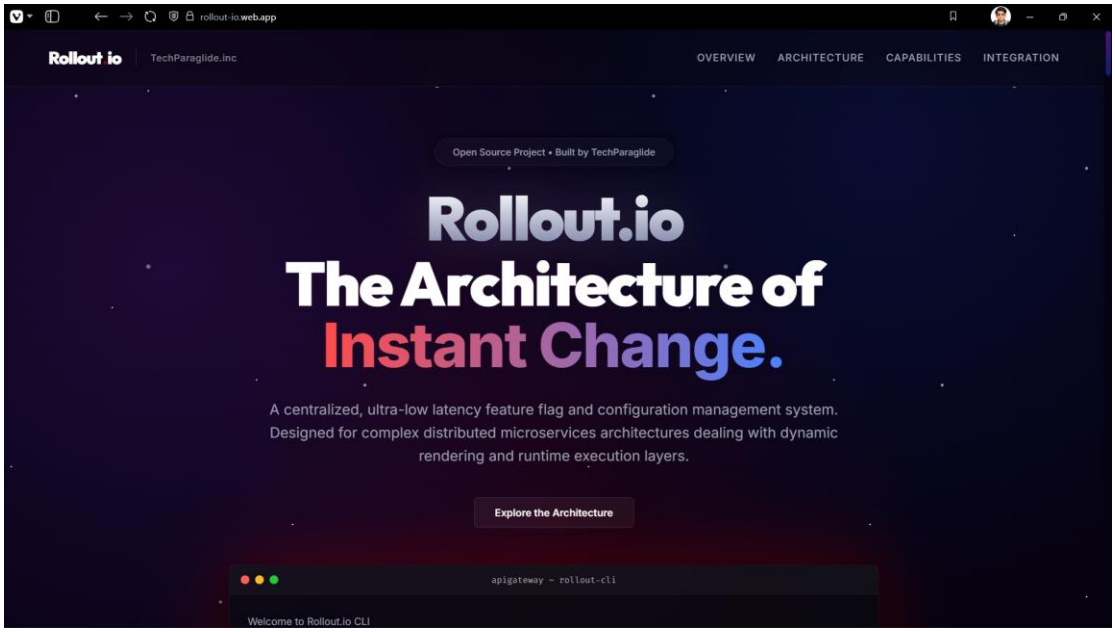

Fig – 6.11: Launch Website Landing Page

Design and Features

The landing page focuses on providing a clean and engaging user experience through:

- Strong visual branding and typography
- Clear headline conveying the platform's purpose
- Informational sections such as Architecture, Capabilities, and Integration
- Call-to-action buttons guiding users to explore the system

The tagline *"The Architecture of Instant Change"* reflects the core objective of enabling dynamic feature control without redeployment.

6.3.2 Admin Dashboard (Control Interface)

The dashboard is designed with a modern user interface, enabling users to easily navigate and perform operations such as creating projects, managing environments, and configuring feature flags.

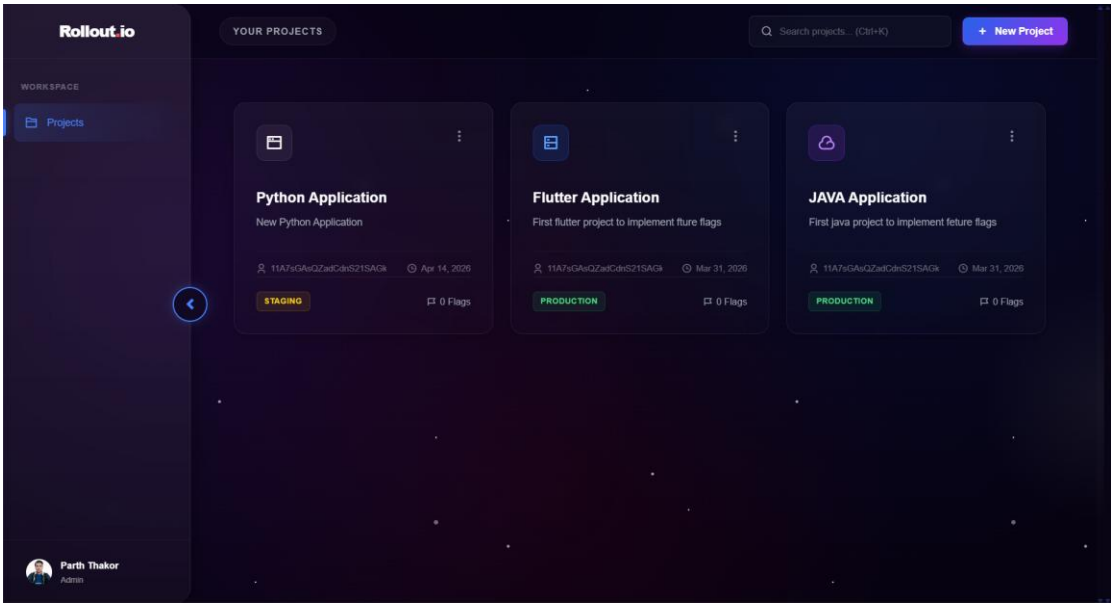

Fig – 6.12: Dashboard Screen

Features of Dashboard

The dashboard provides the following functionalities:

- Creation and management of projects
- Visualization of project details and environments
- Easy navigation through a sidebar-based layout
- Search and filtering of projects
- Quick access to feature flag configurations

6.3.3 Feature Flag Management Interface

The feature flag management screen displays all flags associated with a selected environment and provides controls to enable or disable features dynamically.

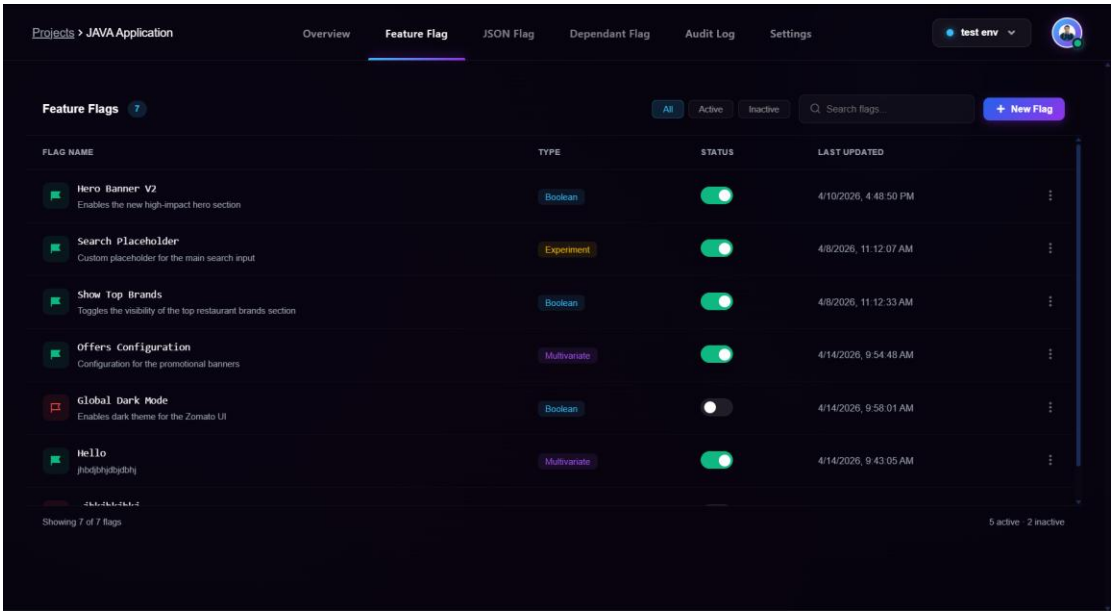

Fig – 6.13: Feature Flag Screen

Key Functionalities

The interface supports:

- Creation of new feature flags
- Enabling and disabling flags using toggle switches
- Viewing flag types such as Boolean, Experiment, and Multivariate
- Monitoring flag status (active/inactive)
- Filtering and searching flags

Dynamic Feature Control

Each feature flag includes a toggle switch that allows users to instantly enable or disable a feature. These changes are reflected in real-time in client applications without requiring redeployment.

Different types of flags allow flexible configurations:

- Boolean flags → Enable/disable features
- Experiment flags → Used for testing variations
- Multivariate flags → Support multiple configurations

6.3.4 SDK Integration and Demo Application (Zomato Clone)

This demo application showcases how feature flags can dynamically control user interface elements and application behaviour in real-time.

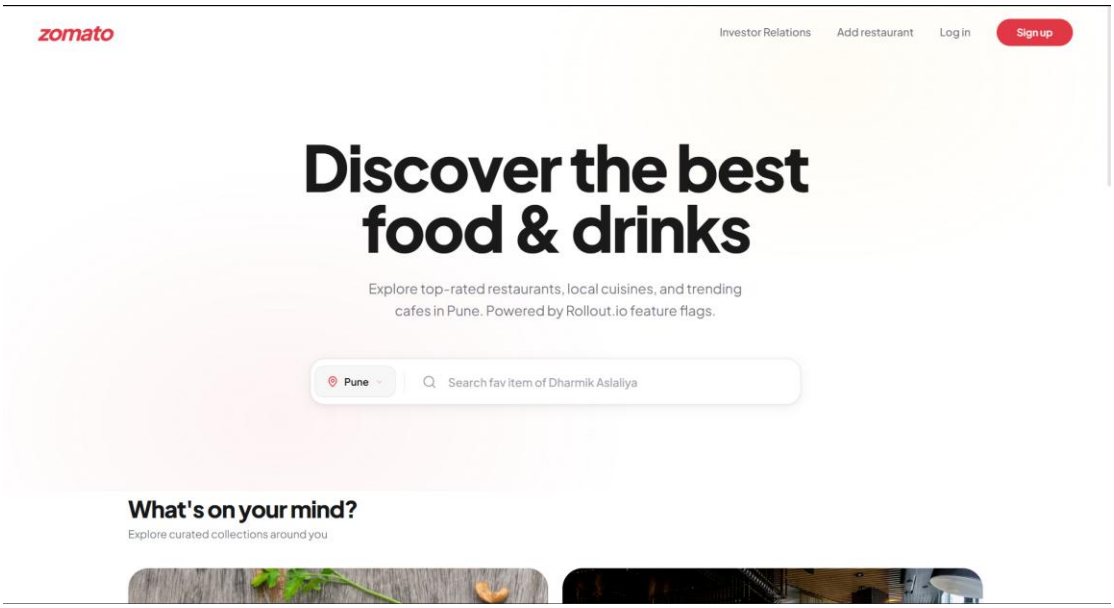

Fig – 6.14: Zomato Clone for Testing

SDK Integration

```
export const RolloutProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [isInitialized, setIsInitialized] = useState(false);
  const [flags, setFlags] = useState<Record<string, any>>({});

  useEffect(() => {
    const init = async () => {
      try {
        console.log(`Initializing official @rollout.io/sdk-js (v${sdk.version})`);

        await sdk.init({
          sdkKey: 'sdk_e5df00b66efd4340a81013e0473e1a58',
          userId: 'user-guest-1',
          baseUrl: 'http://localhost', // Hits ApiGateway on port 80
          refreshInterval: 0 // As per latest request
        });

        console.log("Official SDK Init Success!");
        setIsInitialized(true);
        setFlags(sdk.flags || {});

        // Debugging global
        (window as any).Rollout = sdk;
      } catch (e: any) {
        console.error("SDK Init Error:", e.message);
      }
    };
```

Fig – 6.15: SDK Integration by Developer

Working of SDK

Once initialized, the SDK performs the following operations:

- Fetches feature flags from the backend
- Caches the flags for 30 seconds to improve performance
- Evaluates flags locally within the application
- Sends usage reports back to the server

This ensures low-latency execution and efficient system performance.

Dynamic Feature Control in Demo Application

The Zomato-inspired demo application uses feature flags to control various UI components such as:

- Hero banner visibility
- Search placeholder customization
- Display of top brands section
- Dark mode activation

These features can be toggled from the dashboard, and the changes are reflected instantly in the demo application without requiring redeployment.

6.4 KEY FUNCTIONALITIES IMPLEMENTED

6.4.1 Dependent Feature Flags and Rule-Based Evaluation

The system supports dependent feature flags, where the behaviour of one feature can be controlled based on the state or value of another feature. This enables complex feature rollout strategies and conditional logic within applications.

A visual representation of these dependencies is provided through a graph-based interface, allowing users to understand relationships between different flags.

The above figure illustrates how core flags influence dependent flags through defined relationships. For example:

- A parent feature flag can control the activation of multiple child features

- Conditional logic can be applied using values such as Boolean or numeric thresholds
- Complex rollout strategies can be implemented using dependency chains
- Dependent flag can depend only on core flags to avoid cyclic dependency.

This structure enables developers to create intelligent feature control mechanisms instead of simple on/off toggles.

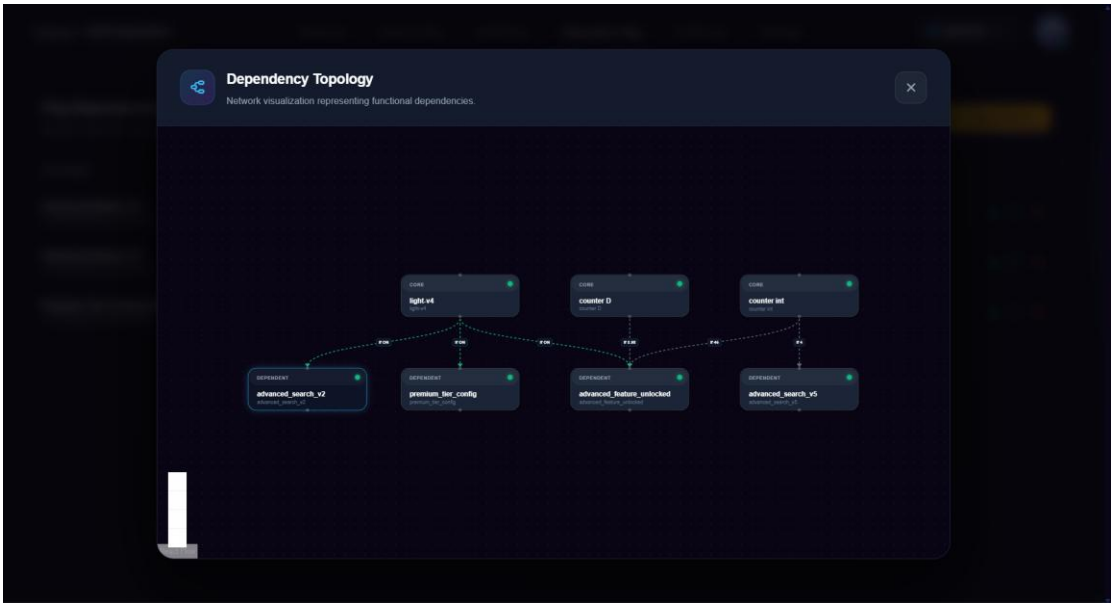

Fig – 6.15: Dependent Flag Graph

6.4.2 JSON-Based Dynamic Configuration

In addition to standard feature toggles, the system supports JSON-based feature flags that allow dynamic configuration of application behaviour and UI.

These flags enable developers to define structured data such as layout, content, and styling, which can be consumed directly by client applications.

The JSON configuration includes parameters such as:

- Title and subtitle content
- Call-to-action (CTA) properties
- UI themes and styles
- Background configurations

These configurations are fetched via the SDK and applied dynamically in the frontend, enabling real-time UI changes without redeployment.

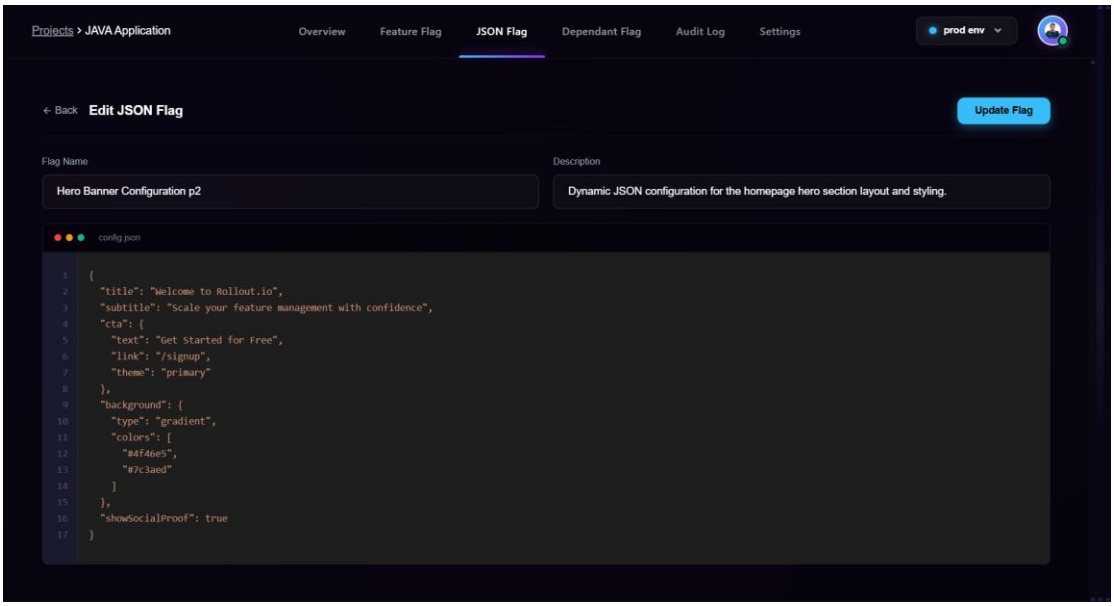

Fig – 6.16: JSON Editor for JSON Flags

6.5 RESULTS AND OUTPUT

The Rollout.io platform was tested through the admin dashboard and integrated client applications to validate its functionality. The system successfully demonstrates real-time feature control, dynamic configuration, and seamless interaction between backend services and frontend applications.

The results confirm that feature flags and configurations can be updated instantly from the dashboard and are reflected in the client application without requiring redeployment.

The following screenshots illustrate the output of the system and validate its real-world usability.

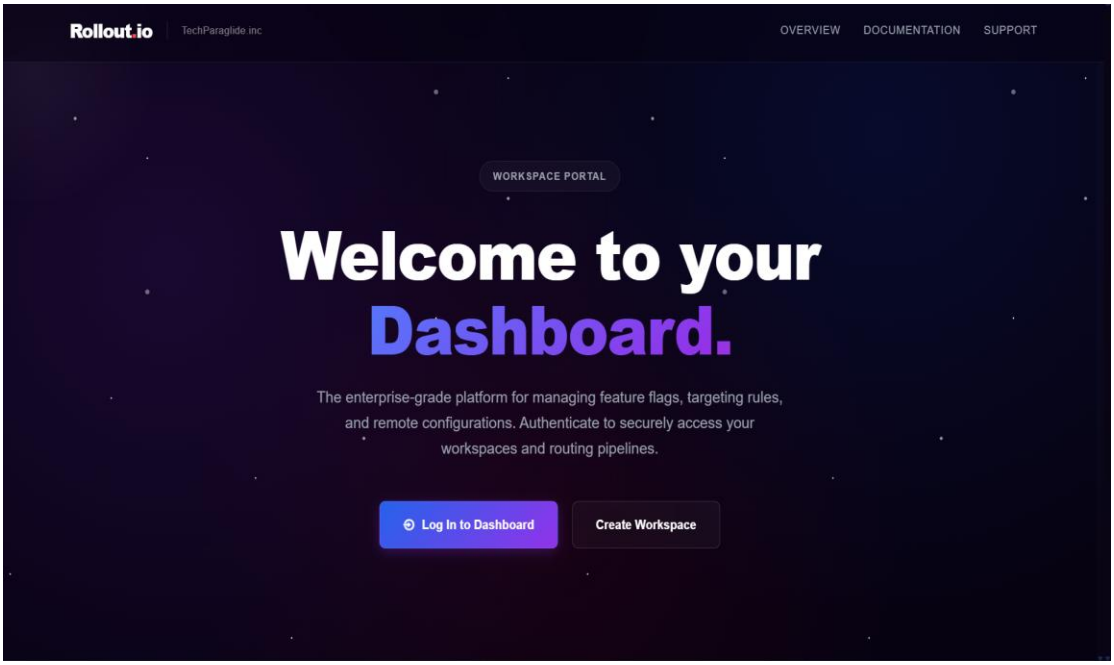

Fig – 6.17: Dashboard Access Interface

This screen represents the entry point to the Rollout.io dashboard, where users can securely access their workspace. It provides options to log in or create a new workspace for managing feature flags and configurations. The interface ensures a smooth transition from landing page to the control dashboard.

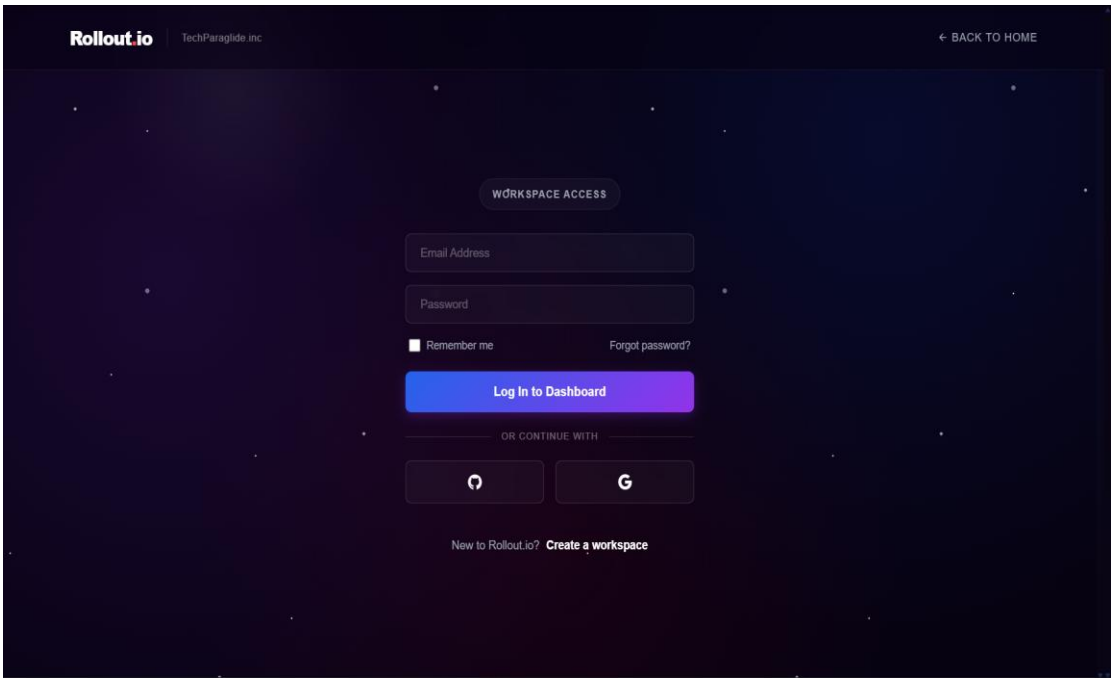

Fig - 6.18: User Authentication Interface

This screen shows the secure login interface for accessing the Rollout.io dashboard. Users can authenticate using email and password or external providers such as GitHub and Google. It ensures controlled access to the platform's feature management system.

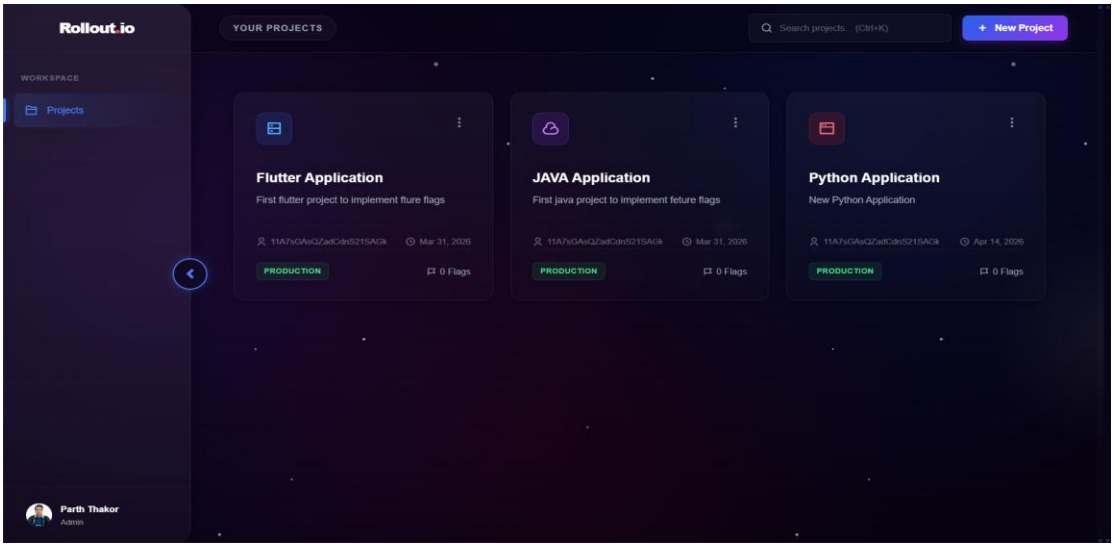

Fig - 6.19: Project Management Dashboard

This screen displays the list of projects within the user's workspace, allowing centralized management of multiple applications. Each project shows its environment status and associated metadata for quick identification. Users can create new projects and navigate into them to manage feature flags and configurations.

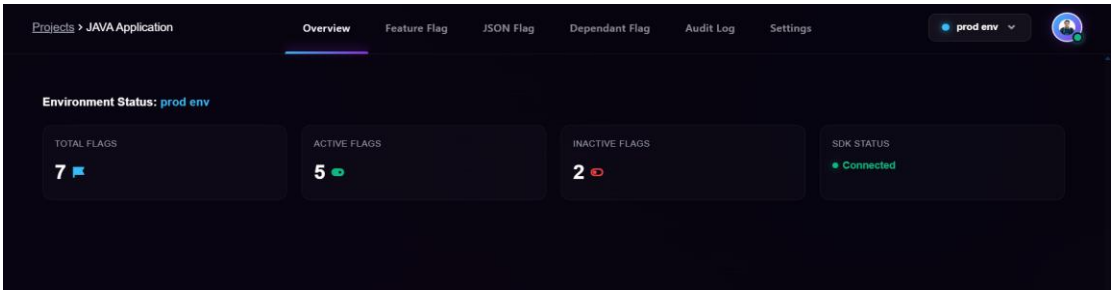

Fig - 6.20: Project Overview and Environment Status

This screen provides an overview of the selected project along with its environment status. It displays key metrics such as total flags, active/inactive flags, and SDK connection status. The dashboard helps monitor system health and feature rollout status in real time.

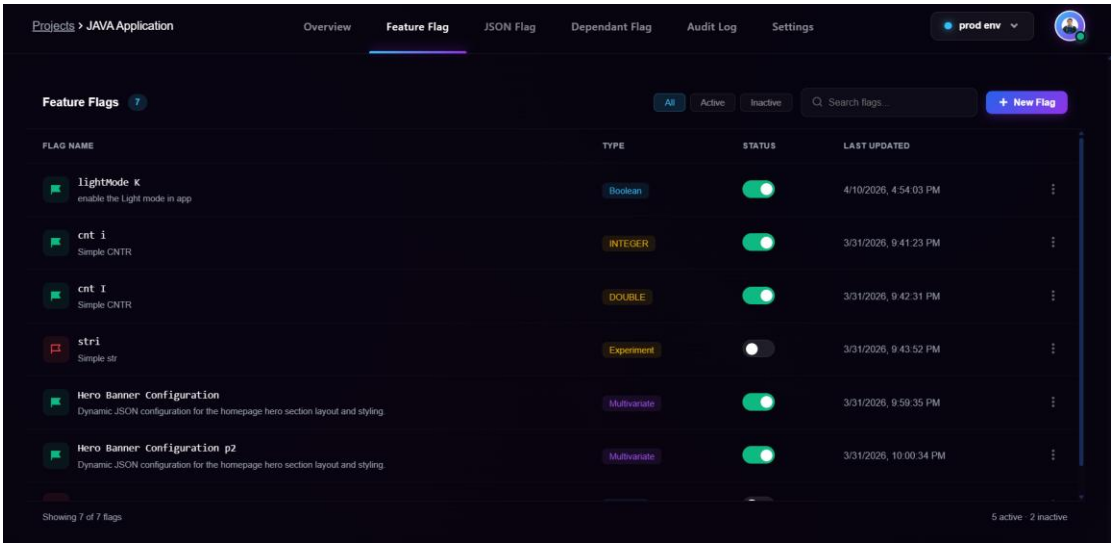

Fig - 6.21: Feature Flags Management Screen

This screen displays all the feature flags created within the selected project. It shows flag types, status (active/inactive), and last updated timestamps. Users can easily enable, disable, and manage flags from this centralized interface.

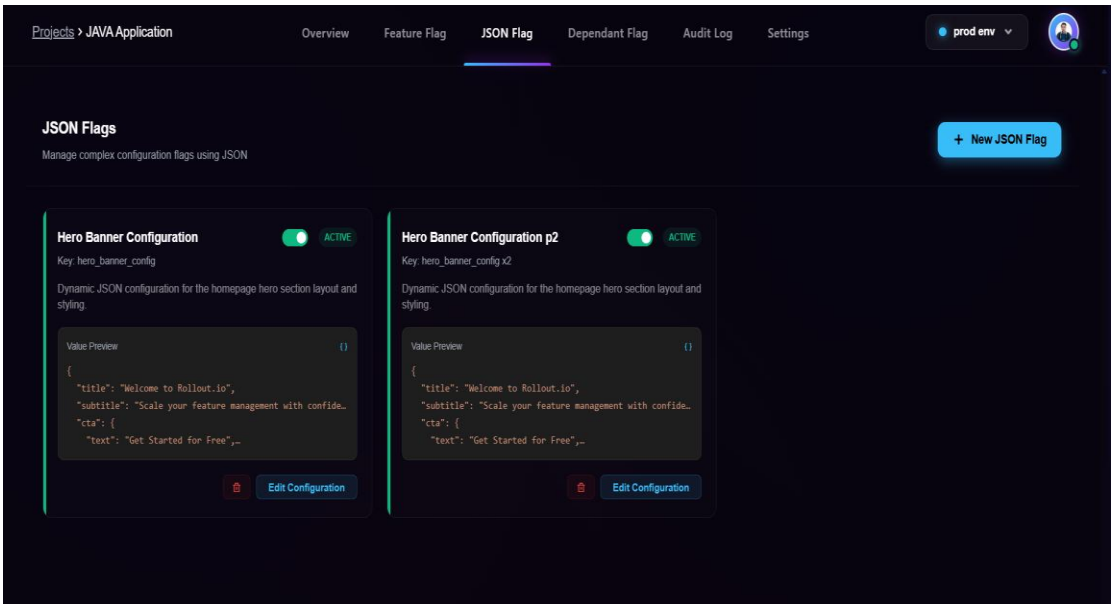

Fig - 6.22: JSON Flags Overview Screen

This screen presents all JSON-based configuration flags in a card layout. Each flag includes a preview of its JSON value and current activation status. It enables easy management of complex dynamic configurations.

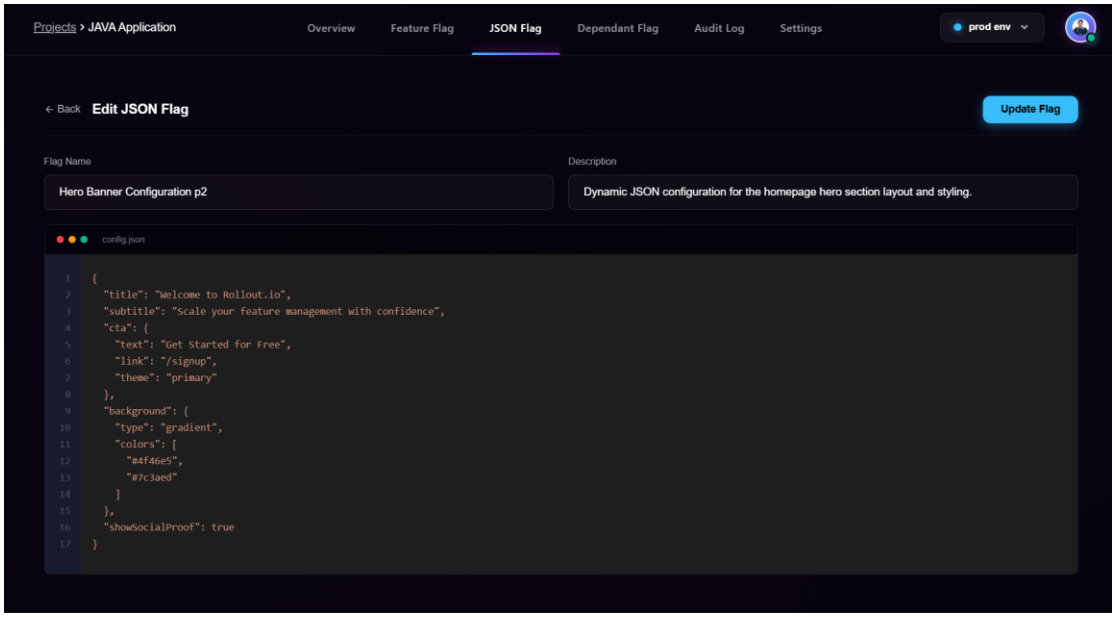

Fig - 6.23: JSON Flag Editor

This screen allows users to create and edit JSON-based feature flags. It provides a structured editor to define dynamic configurations such as UI content and styling. Changes can be updated in real time for instant effect.

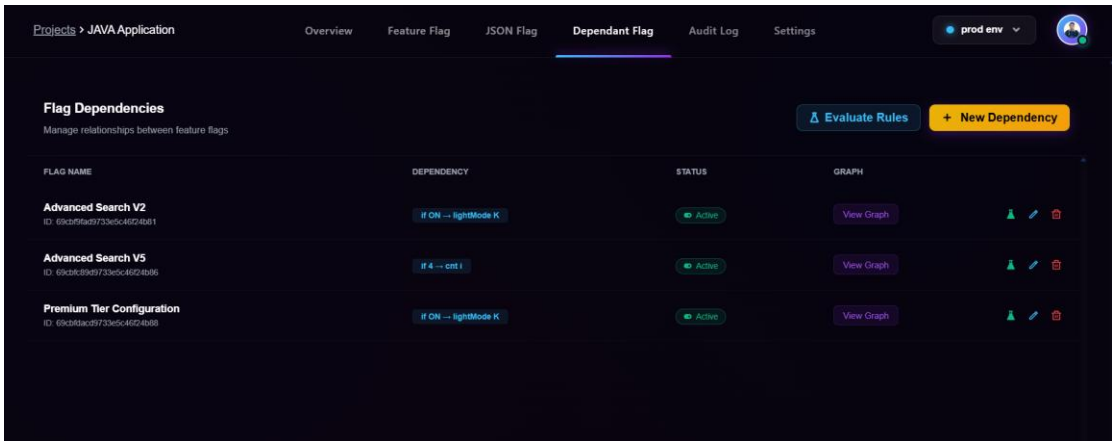

Fig - 6.24: Dependency Rules Management

This screen shows relationships between different feature flags using dependency rules. Users can define conditions such as one flag depending on another. It helps in managing complex feature rollout logic efficiently.

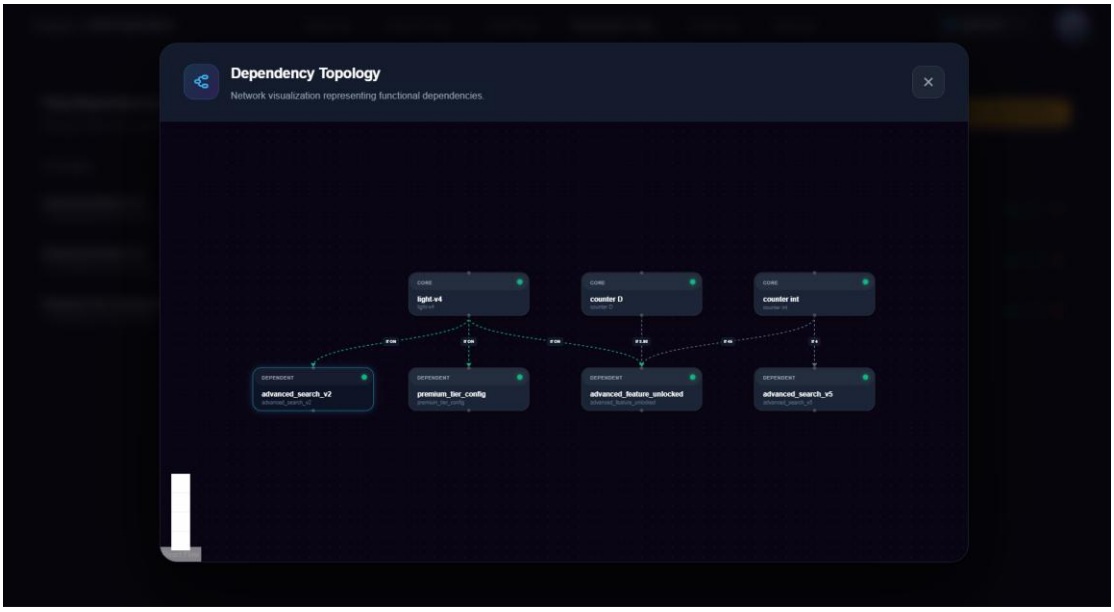

Fig - 6.25: Dependency Topology Visualization

This screen visualizes feature flag dependencies in a graph format. It shows how core flags and dependent flags are connected. This helps developers understand the overall dependency structure clearly.

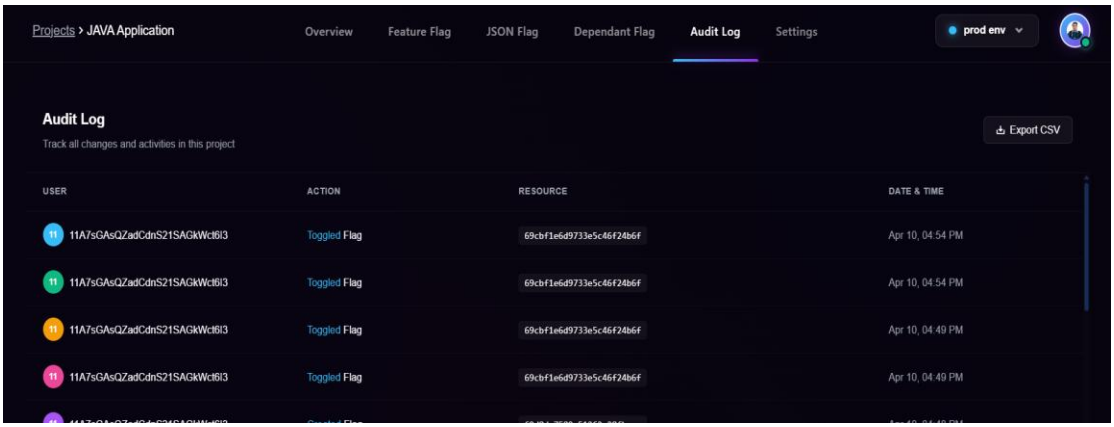

Fig - 6.26: Audit Log Screen

This screen tracks all activities performed on feature flags such as creation, updates, and toggles. Each entry includes user details, action performed, and timestamp. It ensures transparency and traceability of changes.

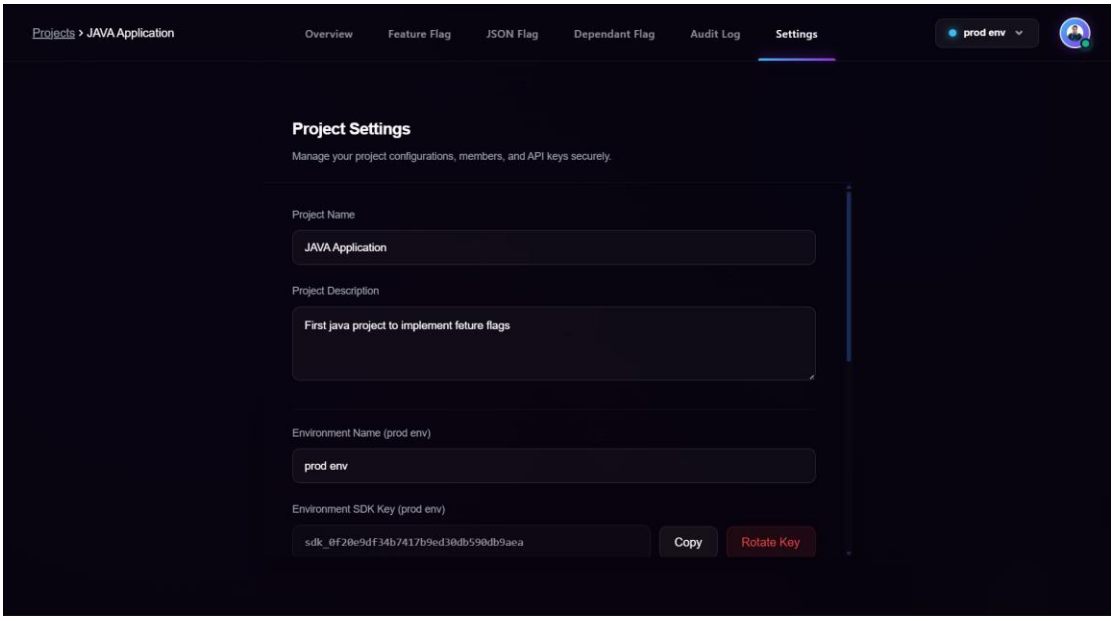

Fig - 6.27: Project Settings Screen

This screen allows users to configure project-level settings. It includes project details, environment configuration, and SDK key management. Users can also rotate and copy SDK keys securely.

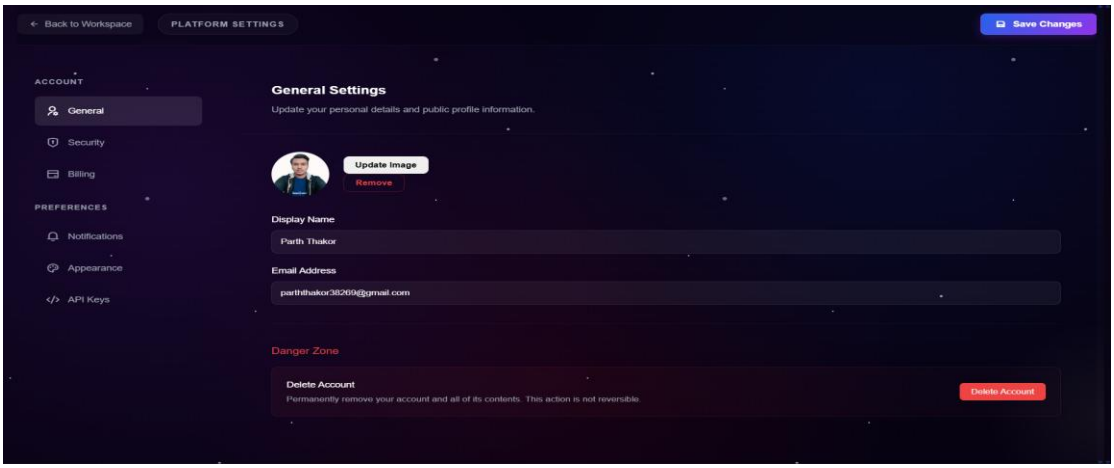

Fig - 6.28: User Account Settings Screen

This screen provides options to manage user profile and account preferences. It includes updating personal details, profile image, and email information. It also contains a secure option to delete the account if needed.

6.6 FUNCTIONALITIES OF THE SYSTEM

Core Functionalities:

- **Feature Flag Management:** Allows users to create, update, enable, and disable feature flags dynamically from the dashboard.
- **Multiple Flag Types Support:** Supports various flag types such as Boolean, Integer, Double, String, and JSON for flexible usage.
- **Targeting Rules:** Enables conditional feature activation based on user attributes and environment settings.
- **JSON Configuration Flags:** Provides support for managing complex configurations using structured JSON data.
- **Dependent Flags:** Allows defining relationships between flags to control feature behaviour based on dependencies.
- **Real-Time SDK Integration:** Ensures that client applications receive updated flag values instantly without restart.
- **Environment-Based Configuration:** Supports separate configurations for different environments like development, staging, and production.
- **Dependency Visualization:** Displays graphical representation of flag dependencies for better understanding.
- **Audit Logging:** Tracks all operations such as flag creation, updates, and toggling with timestamps.
- **SDK Key Management:** Enables secure generation, rotation, and management of SDK keys.

CHAPTER 7: TESTING AND VALIDATION

7.1 INTRODUCTION

Testing is a crucial phase in the development of the Rollout.io system to ensure that all components function correctly and efficiently. This chapter focuses on validating the behaviour of feature flags, API responses, SDK integration, and overall system performance.

The system was tested using different approaches including API-level testing, frontend interaction, and real-time feature flag evaluation. The goal of testing is to verify that the system behaves as expected under various scenarios and that feature updates are reflected correctly without redeployment.

7.2 TESTING APPROACH

API Testing: API testing was performed using Postman to verify the correctness of REST endpoints. Various API requests such as project creation, feature flag management, and SDK flag fetching were executed. The responses were validated to ensure proper structure and expected output.

Frontend Testing: The frontend dashboard was tested to ensure proper user interaction and functionality. Operations such as creating projects, managing environments, and toggling feature flags were verified through the user interface.

SDK Testing: The custom SDK was tested within the demo application to validate real-time feature flag evaluation. The SDK successfully fetched flag values from the backend and applied them dynamically in the application.

Integration Testing: Integration testing was performed to ensure smooth communication between different components of the system. The interaction between

backend services, SDK, and frontend dashboard was validated to confirm end-to-end functionality.

Real-Time Testing: Real-time testing was conducted by updating feature flags from the dashboard and observing immediate changes in the demo application. This verified that the system supports dynamic updates without redeployment.

7.3 REAL-TIME FEATURE FLAG TESTING

7.3.1 Real-Time Feature Toggle (Dark Mode)

To demonstrate the real-time capability of the Rollout.io system, a feature flag controlling the dark mode of the demo application was tested using a before-and-after approach.

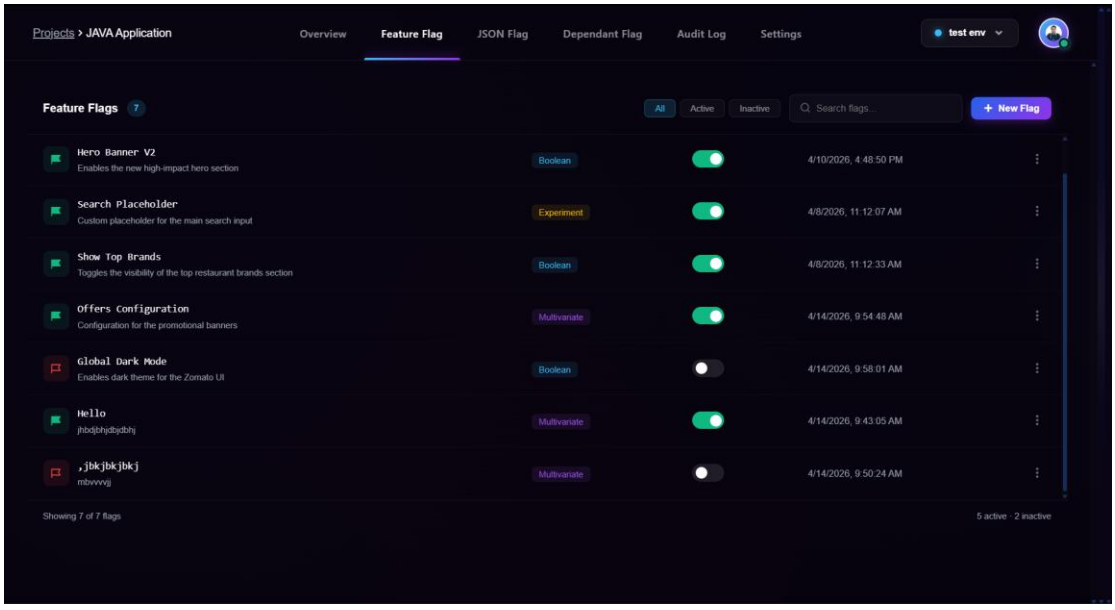

Fig – 7.1: Before Global Dark Mode is Off

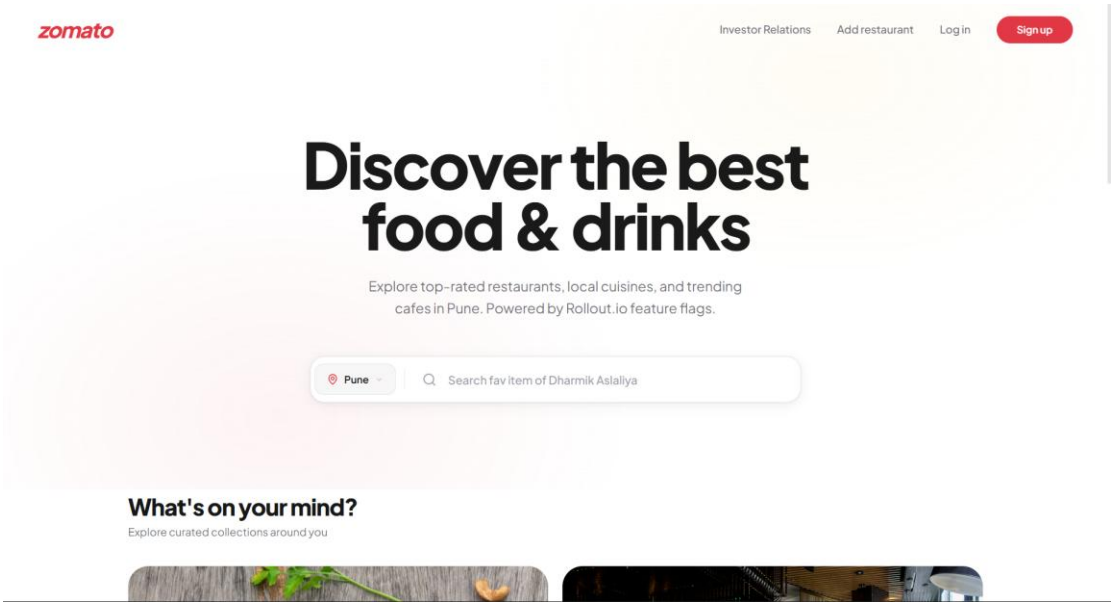

Fig – 7.2: Before light Mode in Test App

Before State (Flag Disabled)

Initially, the *Global Dark Mode* feature flag was disabled from the dashboard. The demo application was displayed in its default light theme, with standard UI colours and layout.

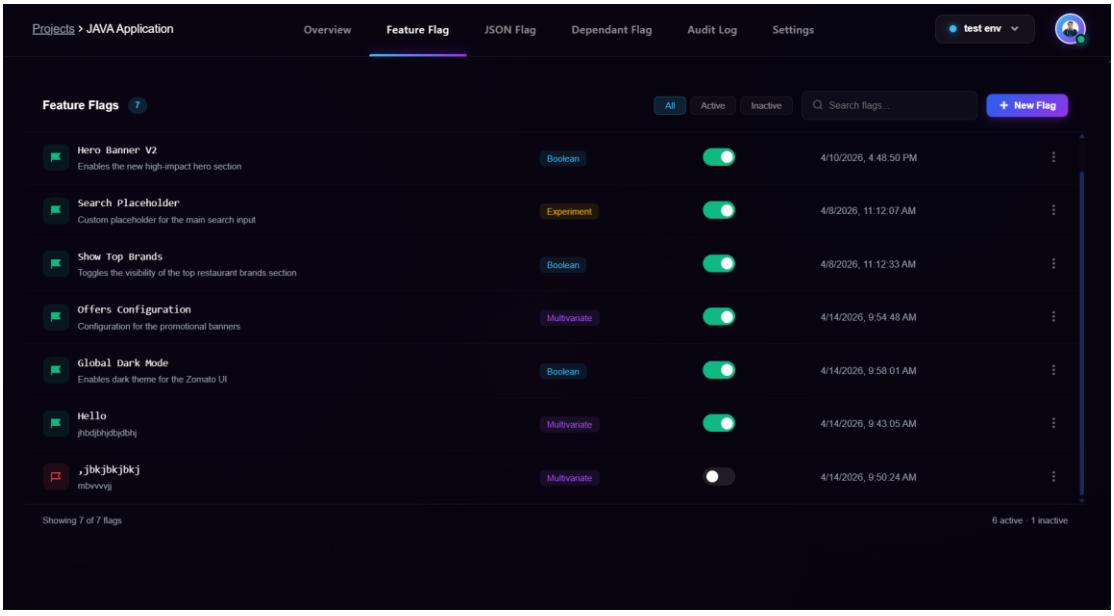

Fig – 7.3: After State of Dashboard Dark mode enabled

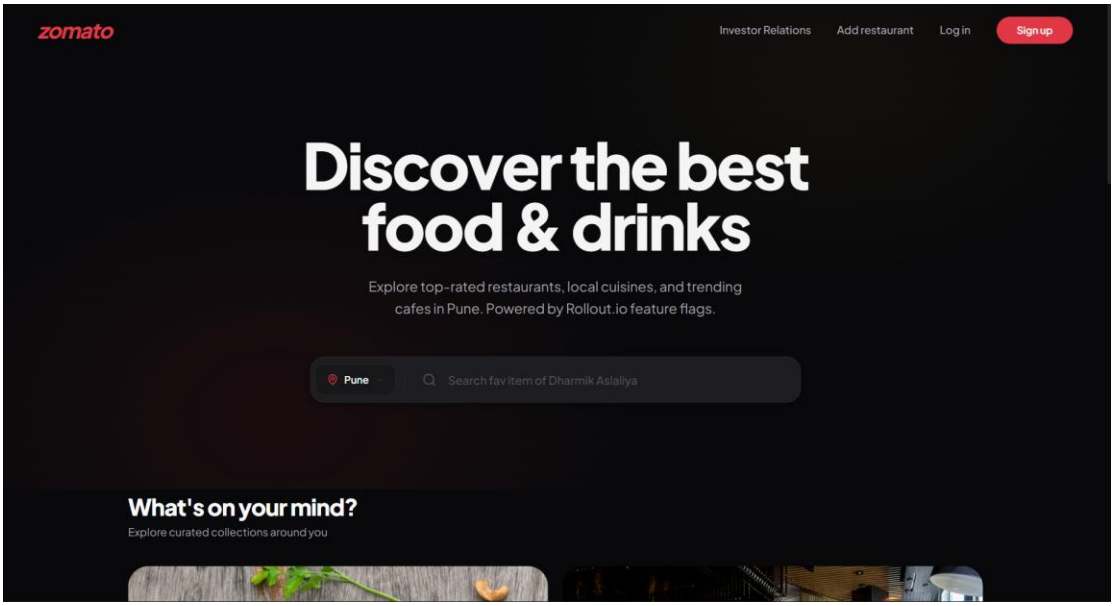

Fig – 7.4: After State in Test Application in Dark Mode

After State (Flag Enabled)

The *Global Dark Mode* feature flag was then enabled from the admin dashboard. The SDK fetched the updated flag value, and the application UI instantly switched to dark mode.

7.3.2 Dynamic Configuration Update (Discount Value Change)

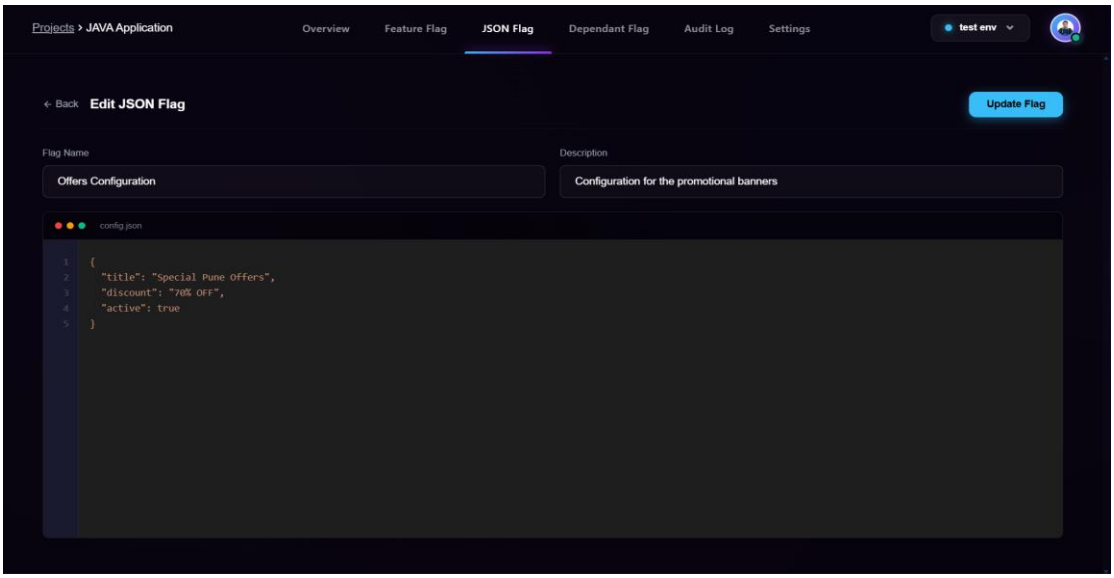

Fig – 7.5: Before state of dashboard

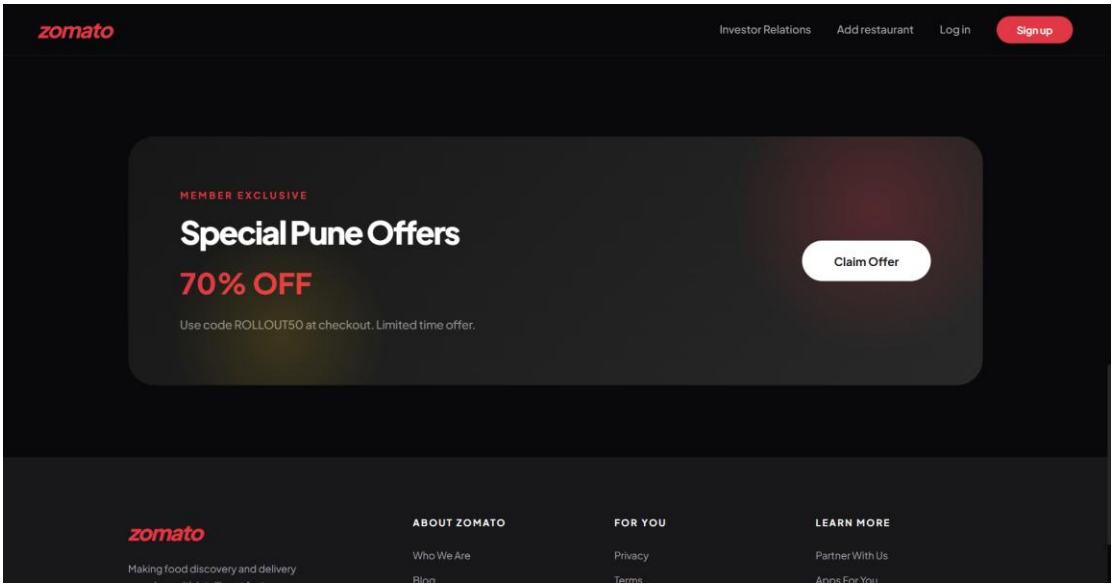

Fig – 7.6: Before state of Test App

Before State (Initial Discount Value)

Initially, the discount value was set to a default number (e.g., 70%) in the JSON configuration. The demo application displayed this value in the user interface.

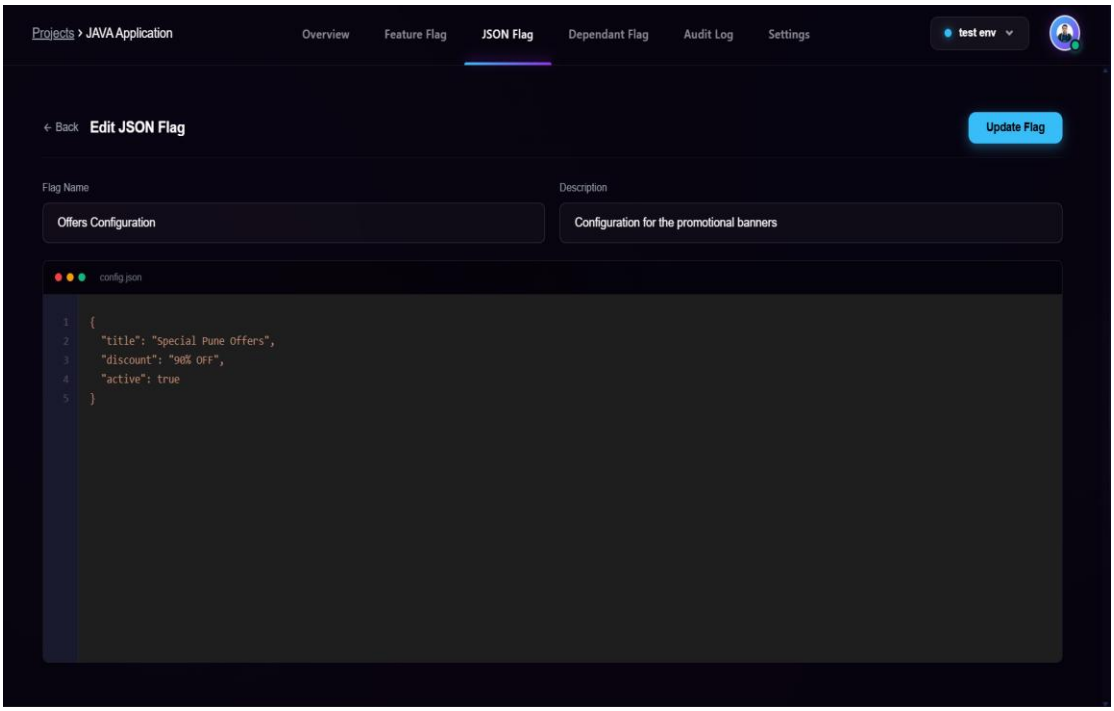

Fig – 7.7: After State of Dashboard

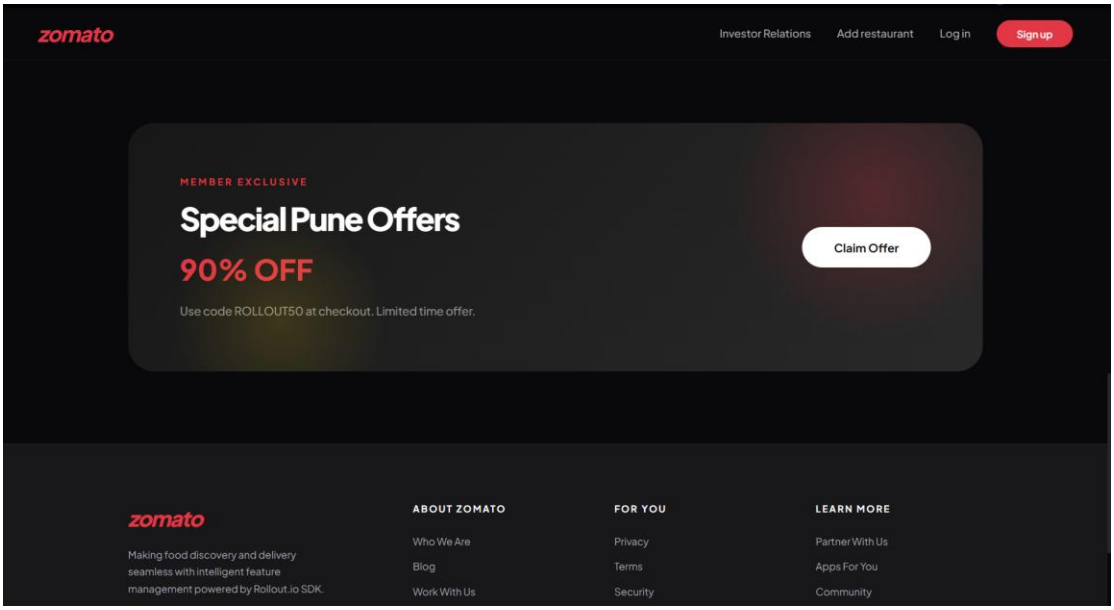

Fig – 7.8: After state of test application

After State (Updated Discount Value)

The discount value was updated from the dashboard by modifying the JSON configuration (e.g., from 70% to 90%). The SDK fetched the updated configuration, and the change was reflected instantly in the application.

7.4 TEST CASES AND RESULTS

The following test cases were executed to validate the functionality and real-time behaviour of the Rollout.io system.

Table 7.1: System Test Cases

Test Case ID	Test Scenario	Action Performed	Expected Result	Actual Result
TC-01	Project Creation	User creates a new project	Project should be created successfully	Project created successfully

TC-02	Feature Flag Creation	User creates a new flag	Flag should be added to system	Flag added successfully
TC-03	Dark Mode (Before)	Dark mode flag is disabled	UI should display light theme	Light mode displayed
TC-04	Dark Mode (After)	Dark mode flag is enabled	UI should switch to dark mode instantly	Dark mode applied in real time
TC-05	Discount Value (Before)	Initial discount value set (e.g., 70%)	UI should display default value	Value displayed correctly
TC-06	Discount Value (After)	Discount updated (e.g., 90%)	UI should update instantly	Value updated successfully
TC-07	JSON Configuration	Update JSON flag content	UI/content should change dynamically	Changes reflected correctly
TC-08	Dependent Flag Logic	Apply dependency between flags	Flag should evaluate based on condition	Dependency logic works correctly
TC-09	Audit Logging	Perform flag update	Action should be recorded in logs	Log recorded successfully
TC-10	Real-Time Update	Modify any flag from dashboard	Changes should reflect without reload	Changes reflected instantly

7.5 LIMITATIONS OF THE SYSTEM

Although the Rollout.io system performs efficiently and provides dynamic feature management capabilities, there are certain limitations observed during testing and implementation.

- The system depends on network availability, as real-time updates require communication between the SDK and backend services.
- Due to SDK caching (30 seconds TTL), immediate updates may not always reflect instantly in some cases.
- Managing complex dependency rules between feature flags can increase configuration complexity.
- The system currently has limited offline support, as feature evaluation relies on backend connectivity.
- Performance may be affected under very high load if not scaled appropriately.

These limitations can be addressed in future improvements to enhance the performance, scalability, and reliability of the system.

CHAPTER 8: CONCLUSION

8.1 OVERVIEW

The Rollout.io platform presents a comprehensive solution for managing feature flags in modern distributed applications. The system is designed to provide real-time control over application behaviour, enabling developers to dynamically manage features without redeployment.

8.2 ACHIEVEMENTS OF THE SYSTEM

The developed system successfully achieves the following:

- Implementation of a microservices-based architecture for scalability and modularity
- Development of a centralized dashboard for managing projects, environments, and feature flags
- Integration of a custom SDK for real-time feature evaluation in client applications
- Support for advanced functionalities such as dependent flags, targeting rules, and JSON-based configurations
- Implementation of audit logging and monitoring for better observability
- Use of Redis caching and Docker containerization for performance and deployment efficiency

8.3 VALIDATION OF OBJECTIVES

The system meets its primary objectives of enabling dynamic feature control and real-time updates. The testing results confirm that feature flags can be updated instantly and reflected in client applications without redeployment, ensuring flexibility and faster development cycles.

8.4 FINAL REMARKS

The Rollout.io platform demonstrates a robust, scalable, and developer-friendly approach to feature flag management. It effectively bridges the gap between development and deployment by allowing safe and controlled feature rollouts.

The system is suitable for real-world applications and can be further enhanced with additional capabilities such as advanced analytics, role-based access control, and global deployment support.

CHAPTER 9: FUTURE SCOPE

9.1 ADVANCED FEATURE ROLLOUT STRATEGIES

Currently, the system supports basic feature toggling and dependency-based control. In the future, more advanced rollout strategies can be implemented, such as:

- **Advance Percentage-Based Rollouts:** Gradually releasing features to a subset of users Segmentation Based (e.g., 10%, 50%, 100%) to reduce risk.
- **A/B Testing and Experimentation:** Running controlled experiments by exposing different feature variations to different user groups and analysing performance.
- **User Segmentation:** Targeting features based on user attributes such as location, device type, or behaviour patterns.

These enhancements will enable data-driven decision-making and safer feature releases.

9.2 REAL-TIME STREAMING AND INSTANT UPDATES

The current system uses polling and caching mechanisms for updating feature flags. This can be further improved by:

- Implementing WebSocket-based communication for real-time updates
- Enabling push-based architecture instead of periodic polling
- Reducing latency for critical feature changes

This will ensure that feature updates are applied instantly across all connected clients.

9.3 ENHANCED SECURITY AND ACCESS CONTROL

To improve system security and enterprise readiness, the following enhancements can be introduced:

- **Role-Based Access Control (RBAC):** Different roles such as admin, developer, and viewer with specific permissions.

- **Fine-Grained Permissions:** Restrict access to specific projects, environments, or feature flags.
- **Audit and Compliance Enhancements:** Advanced logging and compliance tracking for enterprise usage.

These improvements will make the system more secure and suitable for large organizations.

9.4 ADVANCED ANALYTICS AND MONITORING

While the system already integrates monitoring tools, it can be further enhanced by:

- Providing a dedicated analytics dashboard for feature usage
- Tracking user interactions with feature flags
- Visualizing experiment results and performance metrics
- Generating insightful reports for decision-making

This will allow developers to measure the impact of features effectively.

9.5 MULTI-REGION AND DISTRIBUTED DEPLOYMENT

To support global applications, the system can be extended with:

- Multi-region deployment support
- Geo-based routing for low latency
- Data replication across regions

This will ensure high availability and improved performance for users worldwide.

CHAPTER 10: REFERENCES AND RESOURCES

10.1 OFFICIAL DOCUMENTATION

The following official documentation sources were referred to for understanding core technologies:

- Spring Boot Documentation https://spring.io/projects/spring-boot
- Spring Cloud Documentation https://spring.io/projects/spring-cloud
- MongoDB Documentation https://www.mongodb.com/docs/
- Redis Documentation https://redis.io/docs/
- Docker Documentation https://docs.docker.com/
- Prometheus Documentation https://prometheus.io/docs/
- Grafana Documentation https://grafana.com/docs/
- Firebase Authentication Documentation https://firebase.google.com/docs/auth
- OpenAPI (Swagger) Documentation https://swagger.io/docs/

10.2 RESEARCH AND LEARNING RESOURCES

The following platforms and resources were referred to for learning and implementation:

- Technical blogs and tutorials on microservices architecture
- Online forums such as Stack Overflow
- YouTube tutorials for system design and backend development
- Open-source projects related to feature flag systems

These resources helped in understanding real-world implementation patterns and best practices.

10.3 LIVE PROJECT LINK

- **Project Link:** https://rollout.paraglide.in

10.4 SOURCE CODE REPOSITORY

The complete source code of the Rollout.io platform is available in the following GitHub repository:

- **GitHub Repository:** https://github.com/TechParaglide/Rollout.io

The repository includes all backend services, frontend applications, SDK implementation, and deployment configurations.

10.5 SDK PACKAGE (NPM)

The custom JavaScript SDK developed for the platform is published on the NPM registry:

- **NPM Package:** https://www.npmjs.com/package/@rollout.io/sdk-js

This package allows developers to integrate feature flag functionality into their applications easily.

END OF PROJECT

This project titled “Rollout.io – Remote Configuration Platform” has been successfully designed and implemented. **Project Link:** https://rollout.paraglide.in

The system demonstrates real-time feature control, dynamic configuration, and scalable microservices architecture suitable for modern applications.

DEVELOPED BY

PARTHSINH R. THAKOR: BACKEND DEVELOPER

DHARMIK S. ASLALIYA: FRONTEND DEVELOPER

MEET N. PARMAR: FRONTEND DEVELOPER

FINAL YEAR STUDENTS OF BACHELOR OF ENGINEERING

(INFORMATION TECHNOLOGY)